

**Entwicklung einer Android-App zur Erhebung von  
Punktwolken in Straßenzügen**

---

Masterarbeit

Name des Studiengangs  
Angewandte Informatik (M)

**Fachbereich IV**

vorgelegt von  
Emil Schoenawa

Datum:  
Berlin, 21.03.2023

Erstgutachter: Prof. Dr.-Ing. Thomas Jung  
Zweitgutachter: Christoph Holtmann

# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Zielsetzung . . . . .	2
<b>2 Grundlagen</b>	<b>3</b>
2.1 3D-Daten und Tiefenerkennung . . . . .	3
2.1.1 Punktwolken . . . . .	3
2.1.2 Echtzeit-Tiefe . . . . .	4
2.1.3 Fotogrammetrie . . . . .	4
2.1.4 Neural Radiance Field (NeRF) . . . . .	4
2.2 ARCore . . . . .	5
2.2.1 Depth API . . . . .	5
2.2.2 Geospatial API und Visual Positioning Service (VPS) . . . . .	6
2.3 UTM Koordinatensystem . . . . .	6
2.4 Berechnung von Koordinaten aus dem Tiefenbild . . . . .	7
2.5 Berechnung von Geokoordinaten basierend auf einem Offset in kartesischen Koordinaten . . . . .	9
<b>3 Anforderungsanalyse</b>	<b>11</b>
3.1 Stand der Technik . . . . .	11
3.1.1 Smartphones zur Erhebung von 3D-Daten . . . . .	11
3.1.2 Existierende Smartphone Apps zur Erhebung von 3D-Daten . . . . .	12
3.1.3 Erhebung von 3D-Daten in Straßenzügen . . . . .	12
3.2 Anforderungen . . . . .	13
3.2.1 Georeferenzierung und Positionsbestimmung . . . . .	13
3.2.2 Tiefenerkennung . . . . .	14
3.2.3 Weitere Anforderungen . . . . .	14
3.3 Abgrenzung . . . . .	15
<b>4 Umsetzung</b>	<b>16</b>
4.1 Bedienung . . . . .	16
4.1.1 Konfiguration der Geospatial API . . . . .	16
4.1.2 Initialer App-Start . . . . .	16
4.1.3 Scankonfiguration erstellen . . . . .	16
4.1.4 Detailansicht der Scankonfiguration . . . . .	19
4.1.5 Scanvorgang . . . . .	20
4.1.6 UTM Postprocessor . . . . .	22
4.1.7 KML Postprocessor . . . . .	22
4.1.8 Format der Daten . . . . .	23

4.1.9	Import und Visualisierung der Daten . . . . .	25
4.2	Funktionsweise des Scavorgangs . . . . .	27
4.2.1	Prozesse pro Frame . . . . .	27
4.2.2	Prozesse pro Pixel . . . . .	29
4.3	Funktionsweise des UTM Postprocessors . . . . .	30
4.4	Architektur . . . . .	31
4.4.1	Erweiterbare Postprocessor-Architektur . . . . .	31
4.4.2	Wichtige Klassen . . . . .	33
<b>5</b>	<b>Untersuchungen</b>	<b>35</b>
5.1	Grundlagen zur Testdurchführung . . . . .	35
5.1.1	Untersuchung von Kamerapfaden . . . . .	35
5.1.2	Mehrere Scans mit der gleichen Scankonfiguration . . . . .	35
5.1.3	Planarität und Oberflächenvarianz . . . . .	35
5.2	Versuch 1: Scanqualität im Verhältnis zum Abstand . . . . .	37
5.2.1	Durchführung . . . . .	37
5.2.2	Ergebnisse . . . . .	40
5.3	Versuch 2: Horizontale Genauigkeit der Geospatial API . . . . .	45
5.3.1	Durchführung . . . . .	45
5.3.2	Ergebnisse . . . . .	46
5.4	Versuch 3: Übereinstimmung von Scans am gleichen Ort . . . . .	49
5.4.1	Durchführung . . . . .	49
5.4.2	Ergebnisse . . . . .	50
5.5	Versuch 4: Positionsabweichung bei weit entferntem Anker . . . . .	52
5.5.1	Durchführung . . . . .	52
5.5.2	Ergebnisse . . . . .	52
5.6	Sonstige Beobachtungen während der Entwicklung . . . . .	55
<b>6</b>	<b>Diskussion</b>	<b>59</b>
6.1	Zusammenfassung der Ergebnisse . . . . .	59
6.2	Interpretation der Ergebnisse . . . . .	60
6.3	Limitationen der Ergebnisse . . . . .	61
<b>7</b>	<b>Fazit</b>	<b>63</b>
<b>8</b>	<b>Ausblick</b>	<b>65</b>
<b>9</b>	<b>Literaturverzeichnis</b>	<b>67</b>
<b>10</b>	<b>Abkürzungsverzeichnis</b>	<b>73</b>
<b>11</b>	<b>Abbildungsverzeichnis</b>	<b>73</b>



# 1 Einleitung

## 1.1 Motivation

Detaillierte 3D-Daten von Straßenzügen sehen viele Anwendungsfälle in verschiedensten Bereichen. So können durch wiederholte Aufnahme zum Beispiel der Fortschritt von Bauvorhaben, die Nutzung von Parkplätzen oder Änderungen an der Beschaffenheit von Fassaden, Straßen und Gehwegen dokumentiert und untersucht werden. Aber auch zur generellen Visualisierung von Bereichen sind sie geeignet, wenn einzelne Bilder nicht ausreichen.

Die Erhebung von Punktwolken erfolgt meist durch spezielle Laserscanner, LiDAR-Geräte, oder mithilfe von Fotogrammetrie, wobei Bilder und/oder Videos im Nachhinein mithilfe eines Structure-from-Motion (SfM) Algorithmus verarbeitet werden. Letzteres erfordert oft sehr komplexe Algorithmen, die viel Rechenleistung benötigen, während die ersten zwei Lösungen oft spezielle Hardware erfordern, insbesondere, wenn größere Bereiche gescannt werden müssen. Alle Ansätze erfordern teure Investments in Hardware oder monatliche Beiträge zu Cloud-Diensten.

Fortschritte in der Echtzeit-3D-Erkennung von Smartphones [1–3], sowie die Ausstattung einiger Geräte mit Tiefensensoren oder LiDAR-Modulen [4], haben die Entwicklung von Apps ermöglicht, die 3D-Daten aufnehmen können, ohne Nachbearbeitung, wie bei Fotogrammetrie, zu erfordern [5, 6]. Diese Apps bieten eine kostengünstige Alternative zu Ansätzen mit teurer Hardwareinvestition oder monatlichen Kosten für Cloud-Dienste und sind für einige Anwendungsfälle der 3D-Daten vergleichbar gut geeignet.

Die im Jahr 2022 veröffentlichte Geospatial API für ARCore von Google, die Zugriff auf den Visual Positioning Service (VPS) ermöglicht, erlaubt eine deutlich genauere Ortung von Smartphones in urbanen Umgebungen als bisher mit GPS möglich [7]. Bisherige Ansätze zur Georeferenzierung der Punktwolken setzen entweder auf teure Hardware zur genauen Ortung, wie zum Beispiel DGPS-Empfänger, oder auf indirekte Georeferenzierung. Hierbei ist die genaue Geoposition bestimmter Punkte aus Scans bekannt und die gesamte Punktwolke wird so verschoben, dass die Position dieser bekannten Punkte so gut wie möglich mit ihrer realen Position übereinstimmt [8, 9]. Mit der neuen Geospatial API ist es jedoch möglich, eine direkte und genaue Georeferenzierung ohne externe Hardware vorzunehmen.

Zusätzlich wurde ARCores Depth API im Jahr 2022 erweitert, sodass ca.  $65m$  statt etwas mehr als  $8m$  Reichweite unterstützt wird [2]. Dies ermöglicht Echtzeit-Tiefenerkennung in Straßenzügen auf Reichweiten, die bisher selbst mit dedizierter Tiefensorik auf Smartphones nicht möglich war.

## 1.2 Zielsetzung

Im Rahmen dieser Arbeit soll eine prototypische App für Android-Smartphones entwickelt werden, welche die Erhebung georeferenzierter Punktwolken aus Straßenzügen demonstriert. Dabei soll lediglich die Hardware des Smartphones bei der Erstellung der Punktwolken zum Einsatz kommen. Durch die Nutzung der Depth API und der Geospatial API ist so eine breite Kompatibilität mit Android Geräten gewährleistet.

Mit dieser App soll es möglich sein, die Genauigkeit von Googles Geospatial API in Berliner<sup>1</sup> Straßenzügen zu evaluieren. Außerdem soll evaluiert werden, inwiefern sich die Echtzeit-Tiefenerkennung von ARCore (Depth API) zur Erhebung von Punktwolken aus Straßenzügen eignet. Das Ziel möglichst genaue, georeferenzierte Punktwolken zu generieren stellt dabei hohe Genauigkeitsanforderungen an sowohl die Geospatial API als auch die Depth API. Durch die Untersuchungen in dieser Arbeit soll die erwartbare Genauigkeit der entwickelten App bestimmt werden.

Sekundär soll untersucht werden, inwiefern die Genauigkeitsangaben der Geospatial API mit der tatsächlichen Genauigkeit übereinstimmen. Google gab hierfür eine 68-prozentige Wahrscheinlichkeit an [10]<sup>2</sup>.

---

<sup>1</sup>Bedingt durch den Wohnort des Autors. Die Natur vom VPS legt nahe, dass sich die Genauigkeit an anderen Orten unterscheiden kann.

<sup>2</sup>Diese Angabe musste aus einer archivierte Kopie der Dokumentation entnommen werden, da Google seitdem die Angabe nur noch auf die Rotation bezieht.

## 2 Grundlagen

### 2.1 3D-Daten und Tiefenerkennung

3D-Daten bieten eine Vielzahl an Anwendungsfällen, zum Beispiel Trainingssimulationen für autonome Fahrzeuge [11], Vermessung und Datenerhebung in realen Umgebungen [12–14], Daten für visuelles Tracking von Geräten [15], oder bei der Produktion von Filmen [16]. Dementsprechend existieren auch viele Lösungen zur Aufnahme dieser Daten. Grundsätzlich lassen sich die existierenden Lösungen in drei Kategorien einordnen: Echtzeit-Tiefe, Fotogrammetrie und Neural Radiance Fields (NeRF). Die ersten beiden dieser Lösungen erlauben es, Punktwolken zu erstellen<sup>3</sup>.

#### 2.1.1 Punktwolken

Aus Tiefendaten, also einem „Bild“ mit einem Tiefenwert (der Entfernung zur Kamera) pro Pixel (siehe Abbildung 1), können Punktwolken erstellt werden. In Punktwolken besitzt jeder Punkt eine Position, und oft andere Felder, zum Beispiel Farbwerte oder den Abstand zur Kamera. Die Position von Punkten wird mit dem Tiefenwert und der Position des Punktes auf dem Bildschirm, berechnet (vgl. Kapitel 2.4). Damit mehrere Tiefenbilder für die gleiche Punktwolke genutzt werden können, muss die Pose der Kamera für jedes Bild bekannt sein. Dann kann die Position von Punkten im Tiefenbild als Teil des lokalen Koordinatensystems der Kamera angesehen und in ein globales Koordinatensystem umgewandelt werden. Dabei kann die Pose der Kamera entweder innerhalb eines Scans bekannt sein, oder sie ist in einem Referenzkoordinatensystem, zum Beispiel innerhalb eines Raumes, bekannt. Ist die Pose der Kamera auf der Erde bekannt (globale Position und Orientierung), so können die Punkte der Punktwolke globale Koordinaten erhalten. In diesem Fall wird von einer *direkt* georeferenzierten Punktwolke gesprochen [9]. Es ist allerdings auch möglich, und auch oft üblich [9], zur Georeferenzierung von Punktwolken erst die Punktwolke mit einem nicht-globalen Referenzkoordinatensystem zu bestimmen und die Punktwolke im Nachhinein mit globalen Ko-

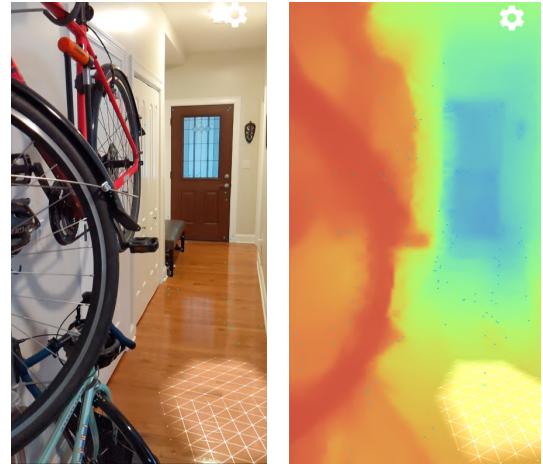


Abbildung 1: Beispiel für ein Tiefenbild (rechts) neben dem normalen Bild (links). Die Tiefenwerte sind durch eine Farbskala visualisiert. Übernommen aus [18].

<sup>3</sup>Es können auch Punktwolken aus NeRFs erstellt werden, allerdings wird das NeRF dabei als reale Szene betrachtet und ein Algorithmus zur stereoskopischen Tiefenerkennung genutzt. Das NeRF ist in dem Falle ein Zwischenschritt. [17]

ordinaten auszustatten. Hierfür wird die globale Position einzelner Punkte der Punktwolke bestimmt, und die gesamte Punktwolke wird anschließend so verschoben, dass die globale Position aller dieser ausgewählten Punkte so gut wie möglich stimmt. In diesem Fall spricht man von *indirekter* Georeferenzierung der Punktwolke [9].

### 2.1.2 Echtzeit-Tiefe

Bei Lösungen zur Erstellung von Punktwolken mit Echtzeit-Tiefe werden stereoskopische Kameras, LiDAR, ToF-Sensoren oder andere Tiefensorik genutzt, um die Tiefe in einem Bild direkt bei der Aufnahme zu bestimmen. Nach dem Scavorgang, oder bei genug Leistung des Geräts auch in Echtzeit, können aus diesen Tiefendaten Punktwolken erstellt werden. In diesem Kontext ist es auch möglich, mehrere schnell aufeinander folgende Bilder aus unterschiedlichen Positionen einer einzelnen, normalen Kamera zu nutzen, um eine Tiefenabschätzung zu erstellen [1]. In diesem Fall sind nur für ausgewählte Frames Tiefendaten vorhanden. Für die fehlenden Frames können die zuletzt bestimmten Tiefendaten jedoch reprojiziert werden, um diese Lücken auszugleichen und eine Echtzeit-Tiefenerkennung „vorzutäuschen“ [1].

### 2.1.3 Fotogrammetrie

Fotogrammetrie hingegen kombiniert eine Vielzahl von Aufnahmen einer einzelnen Kamera an unterschiedlichen Orten zu einer Szene im Nachhinein mithilfe von Structure-from-Motion (SfM) Algorithmen. Dies ähnelt der Funktionsweise der oben erwähnten Tiefenerkennung aus mehreren Frames, ist jedoch mit deutlich mehr Rechenaufwand verbunden und kann erst nach Abschluss eines Scans durchgeführt werden. Fotogrammetrie liefert dafür allerdings auch genauere Tiefendaten [1, 19]. Fotogrammetrie geht oft mit hohen Hardwareanforderungen (z. B. aktuelle Grafikkarten) und/oder hohem Zeitaufwand einher. Die Position der Kamera wird anhand der Änderungen der Perspektive bestimmt und nicht durch Hardware an der Kamera.

### 2.1.4 Neural Radiance Field (NeRF)

Eine relativ neue Entwicklung in dem Bereich von 3D-Daten sind Neural Radiance Fields (NeRF). Diese Art von 3D-Daten erzeugen nicht unbedingt ein Verständnis der Geometrie einer Szene, aber ein Verständnis von der Visualisierung. Somit eignen sich NeRFs sehr gut für Anwendungsfälle, in denen eine Szene visualisiert werden soll, aber nicht so gut, wenn die Geometrie in einer Szene relevant ist. Es ist allerdings auch möglich Fotogrammetrie auf NeRF-Szenen anzuwenden [17] und so die Geometrie einer Szene zu erhalten. NeRFs werden, wie Fotogrammetrie, aus mehreren Einzelaufnahmen durch einen Nachbearbeitungsprozess erstellt und benötigen viel Rechenleistung. Die Kamerapositionen müssen bekannt sein, was jedoch durch einen vorherigen SfM-Nachbearbeitungsschritt möglich ist. Anders als bei herkömmlichen 3D-Daten wird kein Tiefenverständnis der Szene erzeugt, sondern ein neuronales Netz wird mithilfe der existierenden Fotos trainiert, um neue Ansichten mit neuen Blickrichtungen zu erzeugen. [20]

## 2.2 ARCore

Bei ARCore handelt es sich um ein Augmented Reality (AR) Framework von Google. Das Framework ist primär zur Nutzung auf Android Geräten vorgesehen, einige Funktionen werden allerdings auch auf iOS unterstützt. ARCore nutzt eine Mischung aus *Simultaneous Localization And Mapping* (SLAM) und IMU-Daten zur Positionsbestimmung des Smartphones. Im Gegensatz zum Vorgänger *Google Tango* benötigt das Smartphone keinen Sensor zur Tiefenerkennung, um ARCore zu unterstützen.

### 2.2.1 Depth API

Obwohl keine dedizierte Hardware zur Tiefenerkennung vorhanden ist, bietet ARCore seit 2020 eine Tiefenerkennung für das Kamerabild in Form der Depth API [21]. Diese nutzt die Bewegung des Smartphones, um die Tiefe von Objekten im Bild abzuschätzen. Aufgrund der Assoziation der Autoren zu Google, dem Zeitpunkt der Veröffentlichung und der Ähnlichkeit mit der Depth API ist es sehr wahrscheinlich, dass die Depth API auf der in [1] vorgestellten Methodik basiert. In diesem Fall funktioniert die Tiefenerkennung der Depth API so, dass der aktuelle Frame mit einem Frame aus der Vergangenheit zur Tiefenerkennung mittels Stereo-Matching genutzt wird. Zusätzlich kommen ein auf maschinellem Lernen basierter Algorithmus zum Ausschluss von unerwünschten Punkten, sowie bilaterale Filterung zum Einsatz, um das resultierende Tiefenbild zu verbessern und zu verdichten. Da die Tiefenerkennung einige Zeit benötigt, kann in der Zeit, in der die Tiefenerkennung für ein Bild abgeschlossen wird, ein neues Bild für den Nutzer sichtbar sein. Damit die Verzögerung der Tiefenerkennung nicht sichtbar wird, wird das abschließende Tiefenbild durch die Erkennung optischer Kanten verschoben, um es an das neue Bild anzupassen. Valentin et al. bezeichnen dies in [1] als *Late-Stage Slicing*.

Die Depth API kann ein volles Tiefenbild für alle Pixel liefern, wobei viele Pixel durch Glättung und Interpolation leichte Ungenauigkeiten enthalten. Werden genauere Werte benötigt, so bietet die API alternativ ein „rohes“ Tiefenbild (Raw Depth). Hier erhält nicht jeder Pixel einen Tiefenwert, dafür sind die Werte genauer [22]. Für das rohe Tiefenbild wird zusätzlich ein Confidence-Bild bereitgestellt. Dieses repräsentiert die Zuversichtlichkeit der Werte bestimmter Pixel, also wie „sicher“ sich die Depth API bei der angegebenen Tiefe ist.

Im Mai 2022 wurde die maximal erkannte Tiefe der Depth API von  $8,191m$  auf  $65,535m$  erweitert [2]. Zur Genauigkeit können zwei Angaben von Google gefunden werden, zum einen gibt es angeblich die beste Genauigkeit bei Distanzen zwischen einem halben und fünf Metern [18], an anderer Stelle ist von bis zu  $15m$  die Rede [23]. Der Fehler steigt mit der Distanz quadratisch [23].

Die Depth API wird auf 86 Prozent aller aktiven Android-Geräte unterstützt. Die iOS-Implementation von ARCore unterstützt sie hingegen nicht. [24]

### 2.2.2 Geospatial API und Visual Positioning Service (VPS)

Der Visual Positioning Service (VPS) ist ein von Google entwickeltes System zur Positionierung von einem Gerät, oft Smartphones, anhand von den Bildern der Kamera des Geräts. Die Bilder werden mit bekannten Bildern oder Punktwolken eines Ortes verglichen, sodass die aktuelle Position des Smartphones bestimmt werden kann.

Google begann erste Versuche mit seinem VPS mit *Google Tango* in 2017. Es konnten Punktwolken von Innenräumen mit Tango Geräten erhoben werden, damit andere Tango Geräte sich später in diesen Innenräumen lokalisieren können, mit einer Genauigkeit von  $1\text{cm} - 5\text{cm}$ . Der Service war nur als geschlossene Beta verfügbar und wurde nie veröffentlicht, es gab lediglich ein paar Prototypen. [15]

Im Mai 2022 veröffentlichte Google schließlich die Geospatial API, welche Entwicklern Zugang zu Googles auf StreetView-Daten basierenden VPS zur globalen Positionierung von Smartphones gewährt [25]. Dieses System funktioniert überall, wo StreetView-Daten vorhanden sind, inklusive bestimmter Innenbereiche. Ein auf neuronalen Netzen aufgebautes System identifiziert die Teile von StreetView-Bildern, die über längere Zeit erkennbar bleiben (beispielsweise Fassaden). Aus diesen wurde dann eine globale Punktwolke erstellt, gegen die die aktuellen Bilder eines Smartphones verglichen werden können [7]. Der Prozess dieses Vergleichs wird von Google nur als „Computer Vision Algorithmen“ [7] beschrieben, die genaue Funktionsweise ist nicht veröffentlicht. Es ist auch nicht möglich, neue Daten selbst zum VPS hinzuzufügen. Dies bestätigte *Dereck Bridie*, Developer Relations Engineer bei Google [26], auf Anfrage.

Die gelieferte Position der Geospatial API ist eine fusionierte Positionsangabe, die auf sowohl VPS als auch GPS basiert [27]. Ob eine Position aus VPS oder GPS stammt, kann nicht direkt bestimmt werden. Eine solche Angabe „[...] war vor Veröffentlichung einmal Teil der API, wurde aber zugunsten der physikalischen Größe der Ungenauigkeit, entfernt“ [26]<sup>4</sup>. Die Geospatial API liefert, zusätzlich zur Position des Smartphones, auch Genauigkeitsangaben zur Position, Höhe und Orientierung. Diese können genutzt werden, um die Genauigkeit der aktuellen Positionsangabe zu beurteilen.

Die Höhe wird in der Geospatial API als Höhe in Metern über dem WGS-84 Ellipsoiden angegeben [7].

## 2.3 UTM Koordinatensystem

Das *Universal Transverse Mercator* (UTM) Koordinatensystem ist ein Koordinatensystem zur Beschreibung von Positionen auf der Erde mithilfe von Meter-basierten Koordinaten. Hierfür wird die Erde in 60  $6^\circ$ -breite Streifen, sogenannte Zonen, aufgeteilt. Innerhalb jeder Zone können dann Punkte mit zwei Maßen, *Easting* (West nach Ost Position innerhalb der Zone in Metern) und *Northing* (Süd nach Nord Position innerhalb der Zone in Metern), beschrieben werden. Die Zonen sind jeweils durch die Breitengrade  $84^\circ N$  und  $80^\circ S$  begrenzt, da an den Polen die durch die

---

<sup>4</sup>aus dem Englischen übersetzt, Antwort von *Dereck Bridie* auf eine Frage nach einer solchen Angabe durch die Geospatial API in einem privaten Chat

verwendete Projektion erzeugten Fehler zu groß werden. Für Applikation an Zonenübergängen ist Überlappen der Zonen um ca. 40km erlaubt, d.h. eine Zone kann um diese Distanz weiter genutzt werden, auch wenn der neue Bereich eigentlich in einer neuen Zone liegt, um Kontinuität zu gewährleisten. [28]

Der Vorteil dieses Koordinatensystems ist, verglichen mit einem Breiten- und Längengradbasierten, dass die Positionen in Metern angegeben sind und in Kombination mit metrischen Höhenkoordinaten keine Verzerrung der Daten erfolgt. Deshalb eignet es sich gut zur Darstellung von georeferenzierten Punktwolken mit einer Höhenkoordinate in Metern.

## 2.4 Berechnung von Koordinaten aus dem Tiefenbild

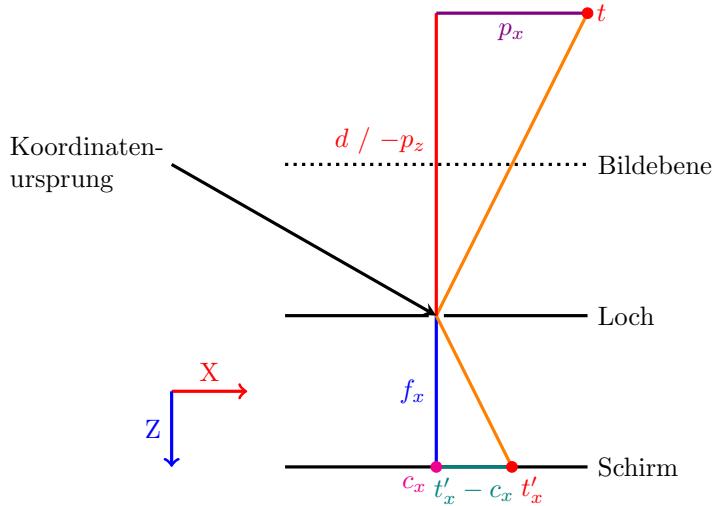


Abbildung 2: Schematische Zeichnung der Berechnung von  $p_x$  aus den intrinsischen Kameraparametern. Das Koordinatensystem der Kamera ist dargestellt mit Achsen und Ursprung. Die Größe  $p_x$  ist die X-Koordinate des Punktes im Koordinatensystem der Kamera,  $c_x$  die X-Komponente des Bildmittelpunktes,  $f_x$  die Brennweite in Pixeln,  $d$  der Tiefenwert für den untersuchten Pixel,  $t$  der aktuell untersuchte Punkt in der realen Welt und  $t'_x$  die X-Koordinate vom Bild des aktuell untersuchten Punktes auf dem Schirm. Es gilt zu beachten, dass das Bild auf dem Schirm nicht gespiegelt ist, im Gegensatz zu einer normalen Lochkamera.

Um eine Punktwolke aus Tiefenbildern zu erstellen, müssen die Szenekoordinaten  $\vec{q}$  der Punkte im Tiefenbild bestimmt werden. Hierfür müssen die Pose der Kamera (Model Matrix  $M$ ), sowie die Position des Punktes im lokalen Koordinatensystem der Kamera ( $\vec{p}$ ) bekannt sein.

Die Position eines Punktes im Koordinatensystem der Kamera kann aus den intrinsischen Parametern der Kamera (bzw. des Tiefenbildes, falls abweichend), dem jeweiligen Tiefenwert, und der Position auf dem Bildschirm bestimmt werden. In ARCore können die intrinsischen Parameter des Tiefenbildes durch Skalierung der intrinsischen Parameter der GPU-Textur mit

der Auflösung des Tiefenbildes erhalten werden [29]. Somit berechnen sich die Brennweite  $f_x$  und  $f_y$ , sowie der Bildmittelpunkt  $(c_x|c_y)$  des Tiefenbildes aus Höhe  $h$  und Breite  $b$  beider Bilder mit:

$$\begin{aligned} f_x &= fgpu_x \cdot \frac{b_{tief}}{b_{gpu}} \\ f_y &= fgpu_y \cdot \frac{h_{tief}}{h_{gpu}} \\ c_x &= cgpu_x \cdot \frac{b_{tief}}{b_{gpu}} \\ c_y &= cgpu_y \cdot \frac{h_{tief}}{h_{gpu}} \end{aligned}$$

Mit den intrinsischen Parametern des Tiefenbildes kann nun die Position eines Punktes im Kamerakoordinatensystem berechnet werden. Wichtig für diese Berechnung ist, ob die Tiefenangabe die Distanz zur Linse der Kamera (Loch im Lochkameramodell) oder zur Ebene der Linse ist. Für die ARCore Depth API ist letzteres der Fall (siehe auch Kapitel 5.6), sodass die z-Koordinate  $p_z$  des Punktes im Kamerakoordinatensystem der negierten Tiefe entspricht. Negiert deshalb, weil die z-Achse des Kamerakoordinatensystems in die Kamera zeigt.

Zur Berechnung der x-Koordinate  $p_x$  kann die Tatsache ausgenutzt werden, dass nach dem Lochkameramodell zwei rechtwinklige Dreiecke vorhanden sind (siehe Abbildung 2). Ein Dreieck definiert durch  $f_x$  und  $t'_x - c_x$ , sowie ein Dreieck definiert durch  $d$  und  $p_x$ . Da beide Dreiecke den gleichen Winkel  $\alpha$  am Loch der Lochkamera haben, können die jeweiligen Verhältnisse der Katheten gleich gesetzt werden:

$$\tan \alpha = \frac{p_x}{d} = \frac{t'_x - c_x}{f_x}$$

Durch Umformung nach  $p_x$  erhält man

$$p_x = d \cdot \frac{t'_x - c_x}{f_x}$$

Die Berechnung der y-Koordinate des Punktes  $p_y$  kann analog erfolgen. Da die y-Achse der Bildschirmkoordinaten jedoch in die entgegengesetzten Richtung der y-Achse des Kamerakoordinatensystems verläuft, muss das Ergebnis negiert werden. Dies kann durch Invertieren der Reihenfolge bei der Differenz zwischen  $c_y$  und  $t'_y$  erfolgen, sodass gilt

$$p_y = d \cdot \frac{c_y - t'_y}{f_y}$$

Zusammenfassend berechnen sich die Koordinaten eines Punktes im Kamerakoordinatensystem also folgendermaßen:

$$\begin{aligned} p_x &= d \cdot \frac{t'_x - c_x}{f_x} \\ p_y &= d \cdot \frac{c_y - t'_y}{f_y} \\ p_z &= -d \end{aligned}$$

Zur Umrechnung in das Szenenkoordinatensystem müssen diese Koordinaten nur noch mit der Model Matrix  $M$ , also der Pose der Kamera, multipliziert werden:

$$\vec{q} = M \times \vec{p}$$

Es gilt zu beachten, dass  $\vec{p}$  hierfür in homogene Koordinaten gewandelt werden muss. Somit sind die Szenekoordinaten  $\vec{q}$  des untersuchten Punktes aus der Position auf dem Tiefenbild, sowie dem Tiefenwert, berechnet.

## 2.5 Berechnung von Geokoordinaten basierend auf einem Offset in kartesischen Koordinaten

Die Geospatial API liefert die Geokoordinaten  $s_{geo}$  bestehend aus  $s_{Breite}$ ,  $s_{Länge}$  und  $s_{Höhe}$  vom Smartphone. Von Punktwolken aus der Depth API sind hingegen nur Szenekoordinaten bekannt, wenn sie nach dem in Kapitel 2.4 beschriebenen Verfahren berechnet werden. Um die Punkte zu georeferenzieren, müssen aus der Geoposition der Kamera  $s_{geo}$  und dem (kartesischen) Offset  $\vec{o}$  zwischen Kamera und einem gegebenen Punkt im Szenenkoordinatensystem, die Geokoordinaten des Punktes ( $p_{geo}$ ) berechnet werden. Hierbei kann vorausgesetzt werden, dass das Szenekoordinatensystem und damit das Offset  $\vec{o}$  bereits nach dem East-Up-South-Koordinatensystem orientiert sind (X zeigt nach Osten, Y nach oben und Z nach Süden), welches von der Geospatial API zur Orientierung von globalen Posen genutzt wird [30].

Die Software *MATLAB* definiert den durchschnittlichen Erdradius in [31] als

$$r_{erde} \approx 6371000m$$

Dieser Wert soll zur groben Berechnung der Offsets genutzt werden. Auf kurzen Distanzen ist der Fehler durch die Annahme einer perfekt runden Erde mit diesem Radius relativ gering, und die Depth API kann maximal Punkte in 65m Entfernung erkennen.

Mithilfe des Erdradius  $r_{erde}$  können nun die Unterschiede in Längen- und Breitengrad anhand der Position des Smartphones ( $\vec{s}$ ) und des Punktes ( $\vec{p}$ ) im Szenekoordinatensystem bestimmt werden:

$$\begin{aligned}\vec{o} &= \vec{p} - \vec{s} \\ \Delta \text{Breitengrad} &= -\frac{o_z}{r_{erde}} \\ \Delta \text{Längengrad} &= \frac{o_x}{r_{erde} \cdot \cos s_{Breite}}\end{aligned}$$

Dabei wird der Wert für den Breitengrad negiert, da die z-Achse des East-Up-South-Koordinatensystem nach Süden zeigt, während die Werte für Breitengrade nach Norden steigen.

Nun kann die georeferenzierte Position  $p_{geo}$ , bestehend aus  $p_{Breite}$ ,  $p_{Länge}$  und  $p_{Höhe}$ , des Punktes aus der Geoposition des Smartphones und den Offsets in Längen- und Breitengrad berechnet werden:

$$\begin{aligned}p_{Breite} &= s_{Breite} + \Delta \text{Breitengrad} \\ p_{Länge} &= s_{Länge} + \Delta \text{Längengrad} \\ p_{Höhe} &= s_{Höhe} + o_y\end{aligned}$$

Somit wurde die Geoposition eines Punktes, anhand einer gegebenen Geoposition und einem kartesischen Offset, bestimmt.

## 3 Anforderungsanalyse

### 3.1 Stand der Technik

Zur Erhebung von 3D-Daten existieren bereits eine Vielzahl an unterschiedlichen Lösungen, die jeweils auf unterschiedliche Technologien setzen.

#### 3.1.1 Smartphones zur Erhebung von 3D-Daten

In [6] untersuchen Constantino et al., wie gut Punktwolken auf Smartphones mit LiDAR oder ToF-Sensoren in urbanen Szenen erstellt werden können. Hierbei nutzten sie die App *3D Live Scanner Pro*<sup>5</sup> [5] auf einem Android Smartphone mit ToF-Sensor und die App *3D Scanner App* auf einem iPad mit LiDAR. Sie konnten identifizieren, dass bis zu einer Distanz von *3m* Scans mit diesen Sensoren relativ zuverlässig möglich sind, mit Ungenauigkeiten im Rahmen von *1cm* bis *3cm*. Probleme haben sie im relativen Tracking der Smartphones identifiziert, insbesondere, wenn das Gerät, mit dem gescannt wird, bewegt wird. Hieraus schließen die Autoren, dass kleine bis mittelgroße Scans gut mit Smartphones mit Tiefensensorik umsetzbar sind. Sie zeigen jedoch auch auf, dass Smartphones mit dieser Technik (noch) nicht besonders verbreitet sind.

Haenel et al. stellen in [3] einen Prototyp zur Erstellung von 3D-Geometrie aus den Daten der ARCore Depth API in Echtzeit vor. Während die Echtzeitanforderung nicht ganz erfüllt wurde, können die Autoren mit *7–8cm* eine relativ hohe Genauigkeit erreichen und stellen Methoden zur Entfernung von Rauschen bzw. Streuung vor. Die Autoren verweisen auf die Kombination aus Genauigkeit, und Echtzeitberechnung, worin die ARCore Depth API vergleichbaren Lösungen überlegen ist. Sie heben im Ausblick explizit die Kombination mit einer Lösung zur Georeferenzierung hervor.

In [32] versuchen Tsoukalos et al. Innenräume mithilfe von Smartphones zu scannen und untersuchen dabei das kommerzielle Unity-Framework *Easy AR*, sowie Googles *ARCore*. Die Autoren ziehen das Fazit, dass der Scan von Innenräumen mit beiden getesteten Methoden unzureichend ist. Sie verweisen jedoch ausdrücklich auf neue Smartphone Modelle mit LiDAR-Scannern, sowie die ARCore Depth API. In ihren Untersuchungen des ARCore Frameworks haben die Autoren nur die Tracking-Punktwolke von ARCore, anstatt einer Punktwolke aus der Depth API, untersucht, welche deutlich spärlicher ist als Punktwolken, die mithilfe der Depth API erstellt werden. Deshalb ist ihre Kritik der zu spärlichen Punktwolke aus dem ARCore Framework auch zu erwarten.

In [12] vergleichen Mikita et al. die 3D-Modelle von Steininformationen aus einem kommerziellen 3D-Scanner mit den Modellen aus Fotogrammetrie mit einer Kamera, Fotogrammetrie auf einem Smartphone (SCANN3D App [33]) und Scans der ARCore Depth API mithilfe der 3D Live Scanner App [5]. Die Autoren sehen Anwendbarkeit aller getesteten Methoden zur 3D-Modellierung

---

<sup>5</sup>Die Pro-Version der App ist inzwischen nicht mehr im Playstore verfügbar. Sie erlaubte die Auswertung von ToF-Sensoren bestimmter Geräte, hauptsächlich der Marken Huawei und Samsung.

ähnlicher Szenen. Sie konnten Ungenauigkeiten bei der Skalierung der Objekte bei Nutzung der SCANN3D App feststellen, welche sie für die Vergleiche manuell ausgeglichen haben.

Spreafico et al. untersuchen in [34], wie gut sich der LiDAR Sensor des *iPad Pro (2020)* für Scans von Gebäuden eignet. Ergebnis der Untersuchungen ist dabei, dass sich das iPad für spezifische Modelle (1:200 Maßstab) nach italienischen Standards eignet. Die Autoren verweisen dabei explizit auf das Distanzlimit des Sensors, welches 5m beträgt.

### 3.1.2 Existierende Smartphone Apps zur Erhebung von 3D-Daten

Für Smartphones existieren eine Reihe an Apps zur Erhebung von 3D-Daten. Die Apps *Reality Scan* [35] und *Polycam* [36] kombinieren Aufnahmen des Smartphones durch einen Fotogrammetrieprozess in der Cloud. Sie ermöglichen die Erzeugung von 3D-Daten mithilfe von Fotogrammetrie, ohne dass der Nutzer dedizierte Hardware für diesen Prozess zur Verfügung haben muss.

Die Apps *Polycam*<sup>6</sup> [36] und *3d Scanner App* [37] auf iOS erlauben die Nutzung des auf einigen iOS Geräten vorhandenen LiDAR zur Erstellung von 3D-Modellen.

Die App *3D Live Scanner* [5] für Android erlaubt die Erstellung von 3D-Modellen anhand der ARCore Depth API [1, 18] und benötigt deshalb keine Cloud zur Nachbearbeitung oder spezielle Sensorik an dem Gerät. Die Ergebnisse sind aufgrund der lokalen Verarbeitung auch schneller verfügbar als vergleichbare Fotogrammetriellösungen. Aufgrund der Kompatibilität der Depth API mit Hardware Tiefensorik [18], wie beispielsweise einem ToF-Sensor, funktioniert diese App auch mit Android Geräten, die einen von ARCore unterstützten ToF-Sensor besitzen. Die Sensordaten werden von der Depth API allerdings nicht direkt genutzt, sondern mit dem Modell der Depth API kombiniert [18].

Die iOS- und Web-App *Luma AI* [38, 39] erlaubt die Erstellung von NeRFs aus einer Reihe an einzelnen Fotos. Die Applikation bearbeitet die Aufnahmen in der Cloud, da die Erstellung von NeRFs das Trainieren eines großen neuronalen Netzes erfordert [20].

### 3.1.3 Erhebung von 3D-Daten in Straßenzügen

In [40] demonstrieren Xiao et al. einen Ansatz zur automatischen Modellierung von Gebäuden anhand von Fotos auf Straßenebene. Besonderer Fokus lag dabei auf der Rekonstruktion der Geometrie von Fassaden aus Referenzbildern. Die Autoren sind in der Lage gute Ergebnisse zu präsentieren, gehen allerdings von rechteckigen Gebäuden aus, was in den Vereinigten Staaten, wo die Tests des Papers durchgeführt wurden, eine valide Annahme ist, aber in europäischen Städten aufgrund der häufig schrägen Dächer, eher weniger anwendbar ist. Dennoch ist diese Methode für eine erste automatische Approximation von 3D-Modellen der Gebäude gut geeignet.

In [41] wird von Pollefey et al. ein System vorgestellt, um aus acht Videostreams in Echtzeit<sup>7</sup> ein 3D-Modell der Umgebung zu erstellen. Hierbei kommt spezielles Aufnahmeequipment

---

<sup>6</sup>Die iOS Version unterstützt LiDAR-Scanning und Photogrammetrie

<sup>7</sup>30Hz möglich, also < 33.3ms für einen Durchlauf der Pipeline

(acht Kameras, Inertial Navigation System (INS) und GPS Empfänger) zum Einsatz, um ein beidseitiges Sichtfeld und eine korrekte Geoposition der Kameras zu ermöglichen. Die Sicht mehrerer Kameras wird von der Pipeline kombiniert, um Tiefeninformationen unter Ausschluss von Hindernissen wie beispielsweise Bäumen für jede Kamera zu gewinnen. Die Tiefeninformationen mehrerer Ansichten werden im Anschluss ebenfalls kombiniert, um Fehler in der Tiefenerkennung auszuschließen. Mit diesem mehrstufigen Prozess sind die Autoren in der Lage, dichte und detaillierte 3D-Modelle von Straßenzügen zu erstellen.

In [42] präsentieren Tancik et al. einen neuartigen Ansatz zur Visualisierung von Stadtumgebungen aus Bildern, die aus einem fahrenden Fahrzeug aufgenommen wurden. Hierbei nutzen die Autoren NeRFs [20] und sind somit in der Lage eine hohe Genauigkeit in der Visualisierung zu erzielen und gleichzeitig Flexibilität bei Umgebungsparametern wie Tageszeit und Wetter zu erlauben. Da NeRFs neuronale Netze zur Visualisierung nutzen, ist bei dieser Lösung keine Geometrie bekannt. Das Koordinatensystem der Visualisierung kann hingegen georeferenziert werden.

## 3.2 Anforderungen

### 3.2.1 Georeferenzierung und Positionsbestimmung

Aus dem aktuellen Stand der Technik wird schnell ersichtlich, dass keine Smartphone-basierten Lösungen zur Erstellung von 3D-Daten direkte Georeferenzierung enthalten. Dies ist auf die mangelnde Genauigkeit von GPS in urbanen Umgebungen zurückzuführen, da GPS bisher der genaueste Weg war, ein Smartphone global ohne zusätzliche Hardware zu lokalisieren. Durch die Einführung der Geospatial API ist nun ein neuer Weg zur Ortung verfügbar, der höhere Genauigkeit als GPS in urbanen Umgebungen bzw. Straßenzügen verspricht, vorausgesetzt die entsprechenden Orte sind mit *Google StreetView* abgedeckt.

Daher soll in dieser Arbeit eine App entwickelt werden, die 3D-Daten in Straßenzügen erhebt und mithilfe der Geospatial API georeferenziert. Dies macht den Schritt der Georeferenzierung im Nachhinein obsolet. Vergangene Arbeiten haben außerdem Ungenauigkeiten beim Positions-tracking der Smartphones hervorgehoben [3, 6, 12, 32], wenn das Smartphone über größere Distanzen bewegt wird. Die Nutzung der Geospatial API kann diesem Problem entgegenwirken. Außerdem existiert das Problem, dass das Szenekoordinatensystem ARCores bei nachfolgenden Scans neu definiert wird, wodurch die Scans im Nachhinein angeglichen werden müssen. Unter Nutzung der Geospatial API hingegen kann die Gerätelocation immer wieder neu im gleichen Koordinatensystem bestimmt werden. Erweiterungen von Scans zu späteren Zeitpunkten, sowie eine genaue Gerätelocation über längere Distanzen, sind mit dieser Positionierungstechnik mit Smartphones möglich, sofern die Genauigkeitsangaben von Google über die API [7] stimmen.

Die globale Georeferenz soll erzielt werden, indem die Koordinaten der 3D-Daten im UTM-Koordinatensystem gegeben sind, sowie mit einer Höhe über dem WGS-84 Ellipsoiden [43]. So sind die Einheiten aller drei Achsen des Koordinatensystems in Metern gegeben und die glo-

bale Position ist bekannt. Die Höhe soll auf den WGS-84-Ellipsoiden bezogen werden, da die Positionsangaben der Geospatial API bereits diese Höhe enthalten. Die jeweilige UTM-Zone soll automatisch (anhand einer Position in den 3D-Daten) bestimmbar sein. Um 3D-Daten an Zonengrenzen korrekt zu behandeln, soll es aber auch möglich sein, die Zone der Daten manuell festzulegen.

### 3.2.2 Tiefenerkennung

Vergangene Arbeiten haben identifiziert, dass dedizierte Sensorik an Smartphones oder Tablets zur Tiefenerkennung, wie ToF-Sensoren oder LiDAR, nur eine Reichweite von bis zu  $5m$  [6, 34] besitzen. Dies genügt nicht für Scans in urbanen Straßenzügen, da die Höhe von Gebäuden Scans aus größerer Entfernung, wie beispielsweise der anderen Straßenseite, erfordern. Außerdem sind Geräte mit dieser Technik noch nicht besonders verbreitet. Insbesondere Android-Geräte, welche noch einen Großteil des deutschen Smartphonemarktes ausmachen [44], besitzen nur selten solche Technik [24].

Fotogrammetrie und NeRFs ermöglichen zwar eine zuverlässige Tiefenerkennung auf größere Distanzen (vorausgesetzt es liegen genug Quellbilder vor), benötigen allerdings auch leistungsfähige Hardware, um ein Ergebnis in absehbarer Zeit zu liefern. Die Generierung auf mobilen Geräten würde zu lange dauern, weshalb PCs mit leistungsstarken GPUs erforderlich sind. Existierende Cloud-Dienste, welche Fotogrammetrie oder NeRF-Generierung anbieten, sind leider nicht von externen Entwicklern um Georeferenz der Quelldaten erweiterbar.

Deshalb soll in dieser Arbeit die Depth API von ARCore zur Tiefenerkennung genutzt werden. Die Erweiterung der API von ca.  $8m$  auf ca.  $65m$  im Jahr 2022 [2] macht sie zu einem für Straßenzüge geeigneten Kandidaten, und die Ergebnisse aus [3] deuten darauf hin, dass die API eine gute Mischung aus Echtzeitverarbeitung und Genauigkeit zur Erstellung von Punktwolken bietet. Sie ist zwar eigentlich nur zur korrekten Okklusion von virtuellen Inhalten in AR-Szenen gedacht [18], bietet aber im Gegensatz zu physischen Sensoren eine größere Reichweite und kann im Gegensatz zu Fotogrammetrie auf Smartphones in Echtzeit genutzt werden [1, 3], ohne dass externe Hardware benötigt wird. Sie wird derzeit von 86 Prozent der aktiven Android-Geräte unterstützt [24].

### 3.2.3 Weitere Anforderungen

Damit die Genauigkeit der Geospatial API separat von der Genauigkeit der Depth API untersucht werden kann, soll der Kamerapfad in UTM-Koordinaten exportiert werden. Er soll außerdem im KML-Format [45] als Linienobjekt vorliegen, damit der Pfad zur Analyse einfach in ein Kartenprogramm wie *Google My Maps* [46] importiert werden kann.

Die 3D-Daten sollen als Punktwolke exportiert werden. So bieten sie die größte Flexibilität für Nachbearbeitung. Die Punktwolke soll dabei im CSV-Format vorliegen, da dieses mit wenig Aufwand erstellt und einfach von anderen Programmen verarbeitet werden kann. Für jeden Punkt sollen Koordinaten, RGB-Werte, der Confidence-Wert, die Entfernung zur Kamera, sowie die

Genauigkeitsangabe der Geospatial API vorliegen. Dies ermöglicht eine detaillierte Analyse der Punktwolke, mit der Möglichkeit, Ausreißer auf spezifische Faktoren zurückzuführen. Zusätzlich soll eine ID vorliegen, welche den Scandurchlauf identifiziert. In weiterführenden Arbeiten können die detaillierten Angaben dann auch zur Verbesserung der resultierenden Punktwolke genutzt werden.

Die mit der App zu erstellenden Scans sollen konfigurierbar sein, um unterschiedliche Schwellwerte für Confidence und Positionsgenauigkeit testen zu können. Es soll auch möglich sein, Scans mit einer initialen Position auszustatten und relativ zu dieser den Scan durchzuführen (ohne kontinuierliche Positionierung).

### 3.3 Abgrenzung

Mit der in dieser Arbeit entwickelten App soll die Erhebung georeferenzierter Punktwolken mithilfe der Geospatial API und der Depth API demonstriert werden. Es sind Ungenauigkeiten durch die Positionierung und durch die Depth API zu erwarten [7, 18]. Die Ungenauigkeiten und ihre Ursachen sollen im Rahmen dieser Arbeit untersucht werden, Algorithmen zur Mitigation dieser Ungenauigkeiten sind jedoch nicht Teil dieser Arbeit.

Mit der App sollen lediglich Punktwolken erstellt werden, welche dann als Grundlage für weitere Projekte dienen können. Die Erstellung von Meshes aus Punktwolken ist ein gut erforschtes Problem mit einer Vielzahl von Lösungen, je nach Anwendungsfall [47], und wird deshalb nicht im Rahmen dieses Projekts behandelt.

Die App muss nicht in Polnähe anwendbar sein. So kann der Umgang mit Koordinaten vereinfacht werden. Aufgrund der sehr geringen Zivilisationsdichte ist diese Limitation akzeptabel. Eine Beschränkung auf die Begrenzungen der normalen UTM-Zonen ( $84^{\circ}N$  und  $80^{\circ}S$ , siehe Kapitel 2.3) soll hier ausreichen.

Zum Export der mit der App erstellten Daten reicht es, die Daten so zugänglich zu machen, dass sie von dem Smartphone auf einen PC heruntergeladen werden können, beispielsweise mit dem Datei-Browser. Eine „Teilen“-Funktion ist nicht erforderlich, und kann bei Bedarf einfach im Nachhinein ergänzt werden [48].

Die App muss lediglich auf Android Geräten funktionieren, die ARCore, die Geospatial API und die Depth API unterstützen. Funktionalität oder korrekte Absturz-Behandlung muss auf nicht unterstützten Geräten nicht garantiert werden. Für eine Liste unterstützter Geräte siehe [24]. Es gilt zu beachten, dass die Funktionalität der App auf Android Versionen über 13 (aktuellste Android Version zum Zeitpunkt der Erstellung dieser Arbeit) nicht garantiert werden kann.

## 4 Umsetzung

Im Rahmen dieser Arbeit wurde die Android-App *Urban Scanner* entwickelt, welche die Depth API und die Geospatial API nutzt, um georeferenzierte Punktwolken zu erstellen. Die Entwicklung erfolgte mit *Android Studio 2021.3.1 Patch 1* in der Programmiersprache *Kotlin*.

Als Entwicklungsgerät kam ein *Google Pixel 6* zum Einsatz. Die App ist jedoch aufgrund des Android-Ökosystems auch auf anderen Android-Smartphones nutzbar, sofern sie die Depth API und Geospatial API unterstützen (siehe [24]).

Der Quellcode der entwickelten App ist dieser Arbeit im .zip-Format angehangen.

### 4.1 Bedienung

#### 4.1.1 Konfiguration der Geospatial API

Damit die App korrekt funktioniert, muss ein API Key für die Geospatial API angegeben werden, bevor die App gebaut wird. Hierfür muss folgende Zeile in die `local.properties` Datei im Wurzelverzeichnis des Projekts eingefügt werden ([API-KEY] durch den echten API Key ersetzen):

```
apiKey=[API-KEY]
```

Um einen API Key zu erhalten, muss erst ein Google Cloud Projekt für die App erzeugt werden. Die dafür notwendigen Schritte sind in [49] erläutert.

#### 4.1.2 Initialer App-Start

Wenn die App das erste Mal geöffnet wird, fragt die App nach Zugriff auf den Standort des Geräts (siehe Abbildung 3a). Dies ist wegen des Berechtigungssystems unter Android erforderlich und damit die App korrekt funktioniert, muss diese Berechtigung gewährt werden. Bei Beginn eines Scavorgangs fragt die App zusätzlich nach Zugriff auf die Kamera. Auch diese Berechtigung muss gewährt werden.

#### 4.1.3 Scankonfiguration erstellen

Jeder Scavorgang beginnt damit, dass eine Scankonfiguration erstellt werden muss. Hierfür muss nach Öffnen der App lediglich die Schaltfläche mit dem „+“-Symbol am unteren Bildschirmrand angetippt werden (siehe Abbildung 3b). Dadurch öffnet sich ein Dialog zum Erstellen der Scankonfiguration (siehe Abbildung 3c).

Das oberste Feld akzeptiert dabei den Namen des Scans. Mit diesem wird der Scan in der Übersicht angezeigt und auch abgespeichert. Eine besondere Validierung ist für dieses Feld nicht implementiert, weshalb für korrekte Funktionalität darauf geachtet werden muss, dass sich der Name als Ordnername eignet und einzigartig ist. Leerzeichen vor und nach dem Namen werden automatisch entfernt.

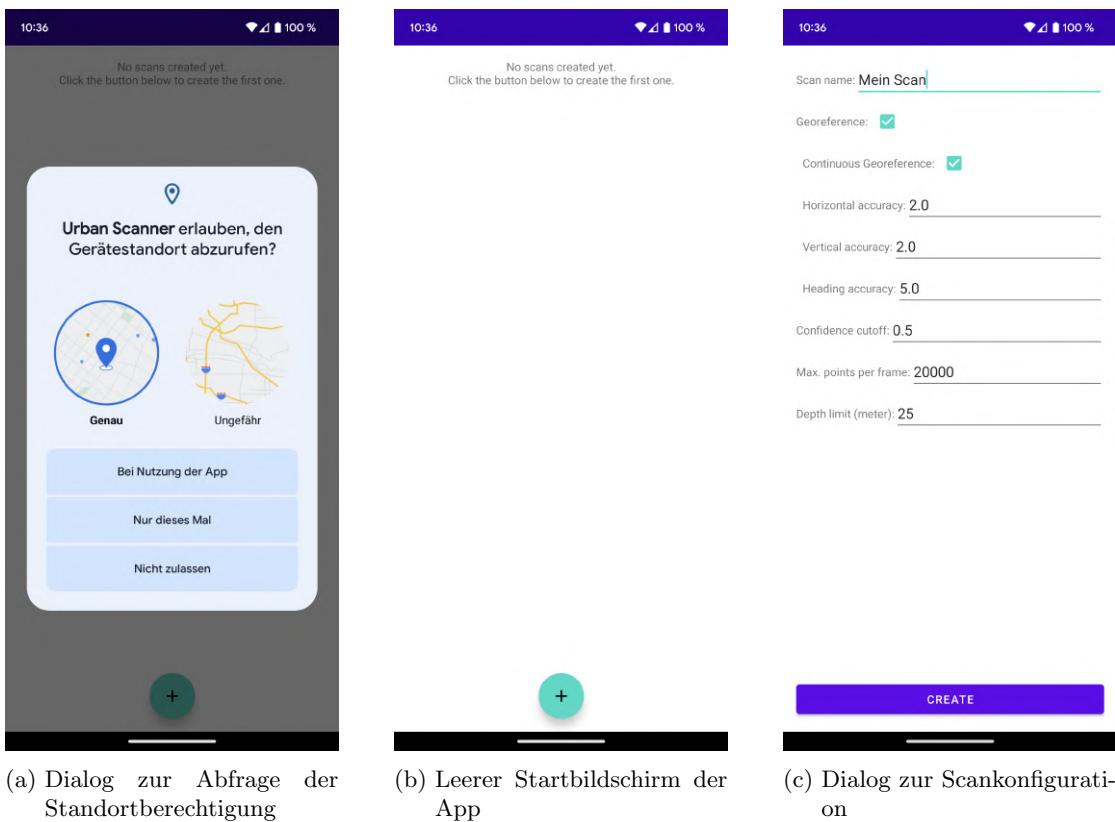


Abbildung 3: Ansichten der App nach erstem App-Start

Der Haken für *Georeference* legt fest, ob die Koordinaten des Scans eine Georeferenz erhalten sollen. Standardmäßig ist dieser Haken aktiviert. Wird der Haken nicht gesetzt, so werden die unter dem Haken eingeordneten Felder (durch Einrückung der jeweiligen Felder erkennbar) ausgegraut, da diese nur mit der Konfiguration der Georeferenzierung zusammenhängen. Scans ohne Georeferenz erhalten nur ein relatives Koordinatensystem. Sie können deshalb auch nach Start eines Scans nicht ergänzt werden, da bei der Ergänzung ein neues Koordinatensystem vorliegen würde.

Der Haken für *Continuous Georeference* legt fest, ob für den Scan immer die aktuelle Smartphone-Position zur Georeferenz genutzt werden soll, oder ob nur der Anker des Scans (also der Startpunkt) eine Georeferenz erhält. Standardmäßig ist dieser Haken aktiviert, es wird also über den gesamten Scan hinweg die aktuelle Position des Smartphones genutzt.

Die Werte für *Horizontal accuracy*, *Vertical accuracy* und *Heading accuracy* legen jeweils die Schwellwerte für horizontale und vertikale Positionsgenauigkeit in Metern und die Genauigkeit der Orientierung in Grad fest. Bei nicht kontinuierlicher Georeferenz müssen diese Schwellwerte von der Geospatial API unterschritten werden, damit der Anker des Scans zu Beginn erstellt werden kann. Bei kontinuierlicher Georeferenz hingegen werden nur Frames ausgewertet, bei

denen die Schwellwerte unterschritten werden. Steigt mindestens einer der Genauigkeitswerte der Geospatial API über die Schwellwerte hinaus, werden so lange keine Frames ausgewertet, bis alle Werte wieder unter den Schwellwerten liegen. Standardwert für die horizontale und vertikale Positionsgenauigkeitsschwelle sind  $2m$ , die Schwelle für Genauigkeit der Rotation liegt standardmäßig bei  $5^\circ$ .

Der Wert für *Confidence Cutoff* legt einen Schwellwert für den Confidence-Wert (siehe Kapitel 2.2.1) der Tiefenpixel fest. Genauer wird hier der normalisierte Confidence-Wert (zwischen 0 und 1) genutzt. Tiefenpixel, die eine geringere Confidence als den Schwellwert haben, werden nicht in die Punktwolke übernommen. Als Standardwert wird hier 0,5 genutzt, da so einige Ausreißer ausgeschlossen werden können (vgl. auch Kapitel 5.6).

Mit *Max. points per frame* kann festgelegt werden, wie viele Punkte maximal pro Frame erzeugt werden. Dieser Wert soll es Nutzern ermöglichen, die pro Frame verarbeiteten Pixel anzupassen, da das Tiefenbild auf unterschiedlichen Geräten unterschiedlich hoch aufgelöst sein kann [50]. Ein hoch aufgelöstes Tiefenbild erfordert, dass für jeden Frame mehr Pixel verarbeitet werden müssen, was wiederum aufgrund der reduzierten FPS einen Einfluss auf die Qualität der Tiefenbilder haben kann (siehe Kapitel 5.6). Mit dem hier eingestellten Wert kann die Pixelmenge manuell begrenzt werden. Das Tiefenbild des in dieser Arbeit genutzten *Google Pixel 6* hat eine Auflösung von  $160 \times 90 = 14400$  Pixel. Diese Pixelmenge kann ohne signifikanten Einfluss auf die Framerate beim Scannen verarbeitet werden, weshalb der Standardwert hier auf 20000 gesetzt wurde, sodass immer alle Pixel ausgewertet werden.

Durch das *Depth limit* kann ein Limit für die Tiefe von Pixeln festgelegt werden. So können, je nachdem aus welcher Entfernung gescannt wird, starke Ausreißer oder unerwünschte Inhalte ausgeschlossen werden. Tiefenpixel, deren Distanz größer als dieses Limit ist, werden nicht bearbeitet. Als Standardwert wurden hier  $25m$  festgelegt, da dies der maximal zuverlässigen Distanz laut der Depth API Dokumentation entspricht [23]. Dennoch wurden im Rahmen dieser Arbeit auch Tests außerhalb dieser Distanz durchgeführt.

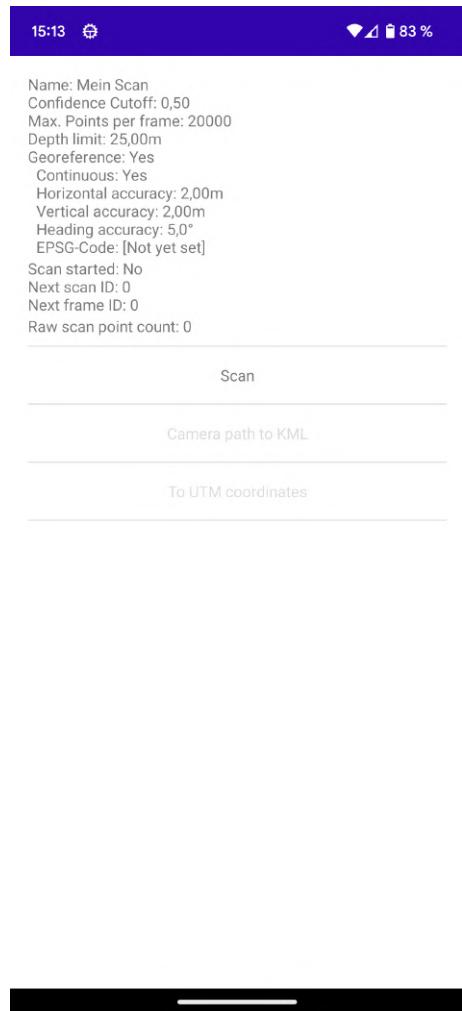


Abbildung 4: Detailansicht einer Scankonfiguration

Nachdem der Scan nach Wunsch konfiguriert wurde, kann durch Antippen der *Create* Schaltfläche die Scankonfiguration erstellt werden. Die Werte der Felder werden hierbei validiert (Wert liegt im richtigen Bereich oder Wert ist eine Zahl) und bei falschen Eingaben wird eine entsprechende Fehlermeldung angezeigt, anstatt dass die Scankonfiguration erstellt wird.

Nachdem die Scankonfiguration erstellt wurde, wird die Detailansicht der Scankonfiguration geöffnet (siehe Abbildung 4). Von hier können die Parameter der Konfiguration erneut eingesehen werden und der Scan kann gestartet werden.

#### **4.1.4 Detailansicht der Scankonfiguration**

Nachdem eine Scankonfiguration erstellt wurde, oder wenn eine Scankonfiguration im Startbildschirm ausgewählt wird, öffnet sich die Detailansicht des Scans (siehe Abbildung 4). Hier kann die Konfiguration des Scans noch einmal eingesehen werden, sowie andere Metainformationen zur Scankonfiguration. So wird hier die nächste Scan-ID angezeigt. Die Scan-ID wird immer dann erhöht, wenn ein Scavorgang beendet wird, sodass mehrere Scans mit derselben Scankonfiguration auseinandergehalten werden können (siehe auch Kapitel 4.1.8). Auch die nächste Frame-ID, sowie die Anzahl an bisher gescannten Punkten, ist in dieser Ansicht einsehbar. Zuletzt wird hier auch der EPSG-Code der verwendeten UTM-Zone angezeigt, sofern der UTM-Postprocessor bereits ausgeführt wurde.

Von der Detailansicht der Scankonfiguration können auch die verfügbaren Nachbearbeitungsprozesse, der UTM-Postprocessor (siehe Kapitel 4.1.6 und 4.3) und der KML-Postprocessor (siehe Kapitel 4.1.7), gestartet werden.

Schließlich wird auch der Scavorgang selbst mithilfe der „Scan“-Schaltfläche gestartet.

#### 4.1.5 Scanvorgang

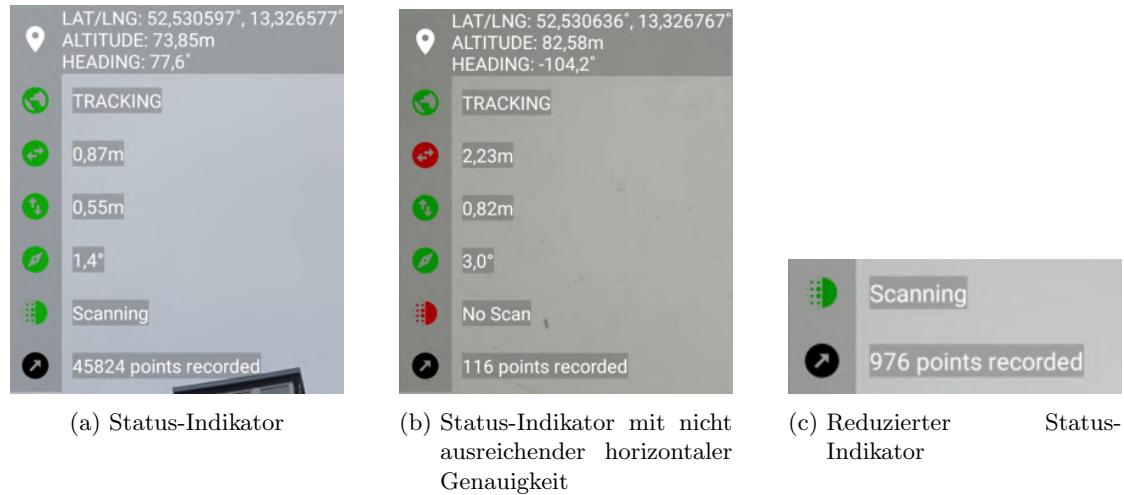


Abbildung 5: Variationen des Status-Indikators

**Status-Indikator** Im Laufe des Scanvorgangs wird in der oberen linken Ecke des Bildschirms immer ein Status-Indikator angezeigt (siehe Abbildung 5a). Ganz oben wird dabei die Geoposition, sowie Orientierung des Smartphones angezeigt, wie von der Geospatial API bestimmt. Darunter repräsentiert ein Globus den Status der Geospatial API. Der Text rechts neben dem Symbol gibt den detaillierten Status an. Das Symbol unter dem Globus (zwei horizontale Pfeile) repräsentiert die horizontale Genauigkeitsangabe der Geospatial API. Ist diese höher als die konfigurierten Schwellwerte, so wird das Symbol in Rot anstatt Grün angezeigt (siehe Abbildung 5b). Die Anzeigen für vertikale Genauigkeit (vertikale Pfeile) und Orientierung (Kompass-Symbol) verhalten sich genauso. Der Wert der jeweiligen Genauigkeitsangabe ist jeweils rechts vom Symbol zu sehen. Dann gibt es noch ein Symbol zur Anzeige, ob innerhalb der letzten Sekunde neue Punkte aufgenommen wurden (Kreis, der zur Hälfte aus Punkten besteht, das vorletzte Symbol). Schließlich zeigt das letzte Symbol (ein diagonaler Pfeil) die Anzahl der bisher gesammelten Punkte an. Der Status-Indikator gibt so einen Überblick darüber, ob gerade neue Punkte aufgenommen werden und ob die Genauigkeitsangaben unter den Schwellwerten liegen. Für Scans ohne Georeferenz wird ein reduzierter Status-Indikator ohne Informationen über die Geräteposition und die Genauigkeitswerte angezeigt (siehe Abbildung 5c).

**Ablauf eines Scans** Wenn ein Scan von der Detailansicht der Scankonfiguration gestartet wird, muss zunächst ein Anker für den Scan platziert werden. Entsprechend zeigt die Schaltfläche am unteren Bildschirmrand ein Anker-Symbol (siehe Abbildung 6a). Wenn ein Scan mit Georeferenz erstellt wird, so müssen die Genauigkeitsschwellwerte bereits bei diesem Schritt erfüllt sein (alle drei entsprechenden Symbole im Status-Indikator grün), ansonsten wird der Anker nicht erstellt und eine entsprechende Statusmeldung wird ausgegeben, wenn die Schaltfläche zur Erstellung

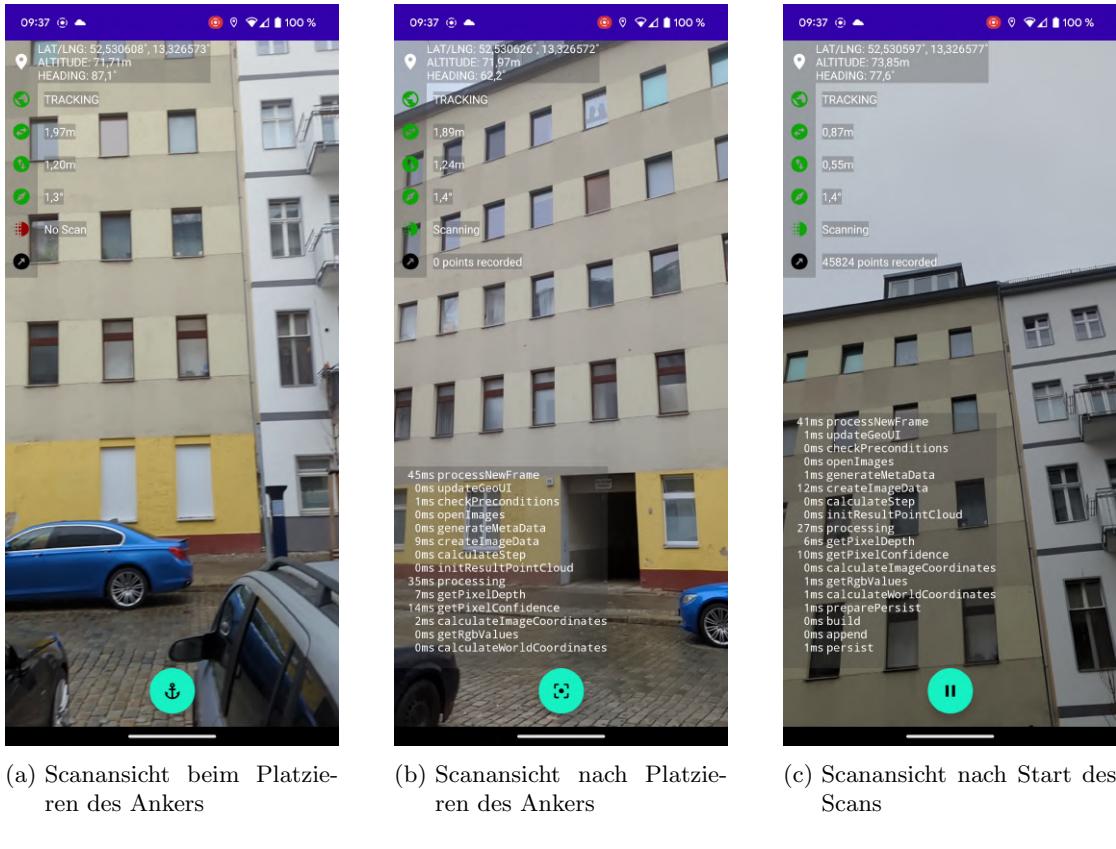


Abbildung 6: Verschiedene Scanansichten

des Ankers angetippt wird.

Nachdem der Anker platziert wurde, kann die Aufnahme von Punkten gestartet werden. Hierfür muss nur die Schaltfläche am unteren Bildschirmrand mit dem Aufnahme-Symbol (siehe Abbildung 6b) angewählt werden. Sobald dies geschehen ist, werden Punkte aufgenommen, solange Tiefendaten vorhanden sind und die konfigurierten Schwellwerte und Limits eingehalten werden. Ob Punkte aufgenommen werden, kann am Scanstatus (zweites Symbol von unten im Status-Indikator) oder an der steigenden Zahl aufgenommener Punkte (unterstes Symbol im Status-Indikator) erkannt werden.

Beim aktiven Scan hat die Schaltfläche am unteren Bildschirmrand ein Pause-Symbol (siehe Abbildung 6c). Dieses kann genutzt werden, um den Scavorgang temporär zu pausieren, zum Beispiel, wenn Personen durch das Bild laufen oder ein Teil einer Fassade nicht gescannt werden soll. Im Anschluss kann der Scan wieder durch dieselbe Schaltfläche fortgesetzt werden.

Während dem Scavorgang (auch wenn der Scan pausiert wurde) werden in der unteren Hälfte des Bildschirms Informationen zur Bearbeitungszeit der einzelnen Frames angezeigt. Diese Informationen sind für den Scavorgang an sich nicht relevant und wurden nur zum Debugging niedriger FPS eingebaut (siehe auch Kapitel 5.6).

Um einen Scanvorgang abzuschließen, kann der Scan pausiert und die „Zurück“-Navigation des Android Geräts genutzt werden, um zur Detailansicht der Scankonfiguration zurückzukehren. Dieser Vorgang erhöht die Scan-ID (siehe Kapitel 4.1.8) um 1. Scans mit Georeferenz (kontinuierlich oder nicht) können anschließend durch erneutes Auswählen der „Scan“-Schaltfläche in der Detailansicht erweitert werden.

#### 4.1.6 UTM Postprocessor

Use custom EPSG (true/false)	<input type="checkbox"/> Optional
EPSG to use (empty for auto)	<input type="checkbox"/> Optional
Add frame metadata (true/false)	<input type="checkbox"/> Optional

Abbildung 7: Konfigurationsmöglichkeiten des UTM-Postprocessors

Nachdem ein georeferenziert Scan durchgeführt wurde, haben die einzelnen Punkte noch keine Geokoordinaten. Diese werden erst mit dem UTM Postprocessor für die einzelnen Punkte ermittelt. Um den UTM Postprocessor für einen Scan durchzuführen, muss zunächst die „To UTM coordinates“-Schaltfläche in der Detailansicht der Scankonfiguration ausgewählt werden. In der nun folgenden Ansicht können einige Optionen für den UTM-Postprocessor festgelegt werden, um den Vorgang zu konfigurieren (siehe Abbildung 7).

Das Textfeld *Use custom EPSG* kann genutzt werden, um die Nutzung einer spezifischen UTM-Zone zu erzwingen. Der EPSG-Code der Zone kann im Feld darunter angegeben werden. Bleibt das oberste Feld hingegen leer oder es wird „false“ eingegeben, wird der EPSG-Code der Zone der Scankonfiguration genutzt (in der Detailansicht der Scankonfiguration sichtbar). Ist kein EPSG-Code gesetzt, so wird die erste Kameraposition eines Scans genutzt, um die zuständige UTM-Zone zu ermitteln. Schließlich kann auch ausgewählt werden, ob die Punktewolkendatei mit UTM-Koordinaten die Metadaten der Frames (Genauigkeitswerte, Scan-ID) enthalten soll (standardmäßig sind diese Felder nicht enthalten, vgl. auch Kapitel 4.1.8). Da alle Konfigurationsfelder optional sind, kann der Postprocessor auch ohne Anpassung direkt gestartet werden.

Ist die Konfiguration erfolgt, kann der Postprocessor durch Auswahl der „Start Processing“-Schaltfläche gestartet werden. Ein Fortschrittsbalken informiert laufend über den Fortschritt des Prozesses. Nach Abschluss navigiert die App zurück zur Detailübersicht der Scankonfiguration. Es gilt zu beachten, dass während der Bearbeitung die App nicht minimiert und das Smartphone nicht gesperrt werden sollte, da die Bearbeitung dann vom Android-System beendet wird, um den Akku zu schonen.

Wird ein Scan nach Abschluss des UTM Postprocessors erweitert, so muss der Postprocessor erneut ausgeführt werden, um die neuen Punkte in die georeferenzierte Punktewolke einzufügen.

#### 4.1.7 KML Postprocessor

Sofern benötigt, kann der für den Scan genutzte Kamerapfad im *Keyhole Markup Language* (KML) Format exportiert werden. So können die Kamerapfade einfach in Programme wie *Goo-*

gle My Maps oder Google Earth importiert werden. Hierfür kann der KML Postprocessor genutzt werden, welcher über die „Camera path to KML“-Schaltfläche in der Detailansicht der Scankonfiguration gestartet werden kann. Es ist keine Konfiguration für diesen Postprocessor erforderlich.

#### 4.1.8 Format der Daten

Jede Scankonfiguration bekommt einen eigenen Ordner in den externen Dateien der App. Diese sind an folgenden Pfaden zu finden und können von dort auf einen PC kopiert werden:

- `/sdcard/Android/data/de.eschoenawa.urbanscanner/files/`
- `/storage/emulated/0/Android/data/de.eschoenawa.urbanscanner/files/`

Der Ordnername entspricht dabei dem Namen der Scankonfiguration. In diesem Ordner befindet sich nach Erstellung der Scankonfiguration zunächst nur die `meta.json` Datei. Diese Datei enthält die Werte der Scankonfiguration (siehe Kapitel 4.1.3 für eine Erklärung einzelnen Felder) und verwaltet veränderliche Metadaten wie die aktuelle Scan-ID, die aktuelle Frame-ID und den EPSG-Code der vom UTM Postprocessor genutzten UTM Zone. Die Datei dient nur zur Verwaltung der Scankonfiguration in der App und ist nicht relevant für den Export von Daten.

Wurde mindestens ein Scan durchgeführt, werden eine `raw.xyz` und eine `frames.csv`-Datei erstellt (siehe auch Tabelle 1). Die Datei `raw.xyz` enthält die Punkte mit Positionen relativ zum Anker des Scans (also nicht georeferenziert), den Confidence-Werten, der Farbe und der dazugehörigen Frame-ID im CSV-Format (ein Punkt pro Zeile). Die Frame-ID erlaubt es, jeden Punkt einem Frame in der `frames.csv`-Datei zuzuordnen. Diese Datei enthält Informationen über die Frames eines Scans, wie die Scan-ID, die Kameraposition relativ zum Anker des Scans, und, falls der Scan georeferenziert ist, die Geoposition der Kamera, inklusive den Genauigkeitsangaben der Geospatial API. Es gilt zu beachten, dass neue Scancvorgänge einen neuen Anker benutzen können, weshalb die relativen Positionen beider Dateien nur für die gleiche Scan-ID den gleichen Anker als Referenz nutzen.

Wenn der UTM Postprocessor ausgeführt wird, werden zwei neue Dateien mit UTM-Koordinaten erstellt. Zum einen die Datei `utm.xyz`, welche die georeferenzierte Punktwolke enthält, zum anderen die Datei `frames_utm.csv`, welche die Kamerapositionen der `frames.csv` in UTM-Koordinaten übersetzt. Die georeferenzierten Punkte in der `utm.xyz` Datei besitzen ihre Position mit UTM-Northing und UTM-Easting, sowie ihre Höhe über dem WGS-84-Ellipsoiden. Dabei wurde Northing auf die x-Achse gelegt und Easting auf die z-Achse, da *CloudCompare* ein rechtshändiges Koordinatensystem nutzt und nur so keine der Achsen gespiegelt werden muss. Außerdem sind die Farbe, die Confidence und die Distanz zur Kamera als Skalarfelder für die Punkte gegeben. Wurde der UTM Postprocessor konfiguriert, um Frame-Metadaten zur Punktwolke hinzuzufügen (vgl. Kapitel 4.1.6), so sind zusätzlich die Genauigkeitsangaben der Kameraposition von der Geospatial API, sowie die Scan-ID als Felder vorhanden (siehe auch Tabelle 1).

Dateiname	Erstellt durch	Format	Felder
raw.xyz	Scan	CSV	relativ X relativ Y relativ Z Confidence (0-1) Rot (0-255) Grün (0-255) Blau (0-255) Frame-ID
frames.csv	Scan	CSV	Frame-ID Scan-ID Kamera relativ X Kamera relativ Y Kamera relativ Z <i>Kamera Breitengrad</i> <i>Kamera Längengrad</i> <i>Kamera Höhe</i> <i>Kamera Orientierung</i> <i>Horizontale Genauigkeit</i> <i>Vertikale Genauigkeit</i> <i>Orientierung Genauigkeit</i>
utm.xyz	UTM Postprocessor	CSV	Northing Höhe Easting Rot (0-255) Grün (0-255) Blau (0-255) Confidence (0-1) Distanz zur Kamera <i>Horizontale Genauigkeit</i> <i>Vertikale Genauigkeit</i> <i>Orientierung Genauigkeit</i> <i>Scan-ID</i>
frames_utm.csv	UTM Postprocessor	CSV	Frame-ID Scan-ID Kamera Northing Kamera Höhe Kamera Easting Horizontale Genauigkeit Vertikale Genauigkeit Orientierung Genauigkeit
frames.kml	KML Postprocessor	KML	(Linien)

Tabelle 1: Dateien einer Scankonfiguration. Optionale Felder sind *kursiv* gekennzeichnet.

Zuletzt ist noch die *frames.kml* Datei zu erwähnen, welche den durch die *frames.csv* Datei beschriebenen Kamerapfad als ein Linienobjekt pro Scan-ID im KML-Format [45] enthält. Die ersten sechs Scan-IDs erhalten dabei unterschiedliche Farben (rot, grün, blau, gelb, türkis, lila), danach wiederholt sich die Farbreihenfolge. Die Linienobjekte bekommen Namen nach dem Schema „Scan [Scan-ID]“.

#### 4.1.9 Import und Visualisierung der Daten

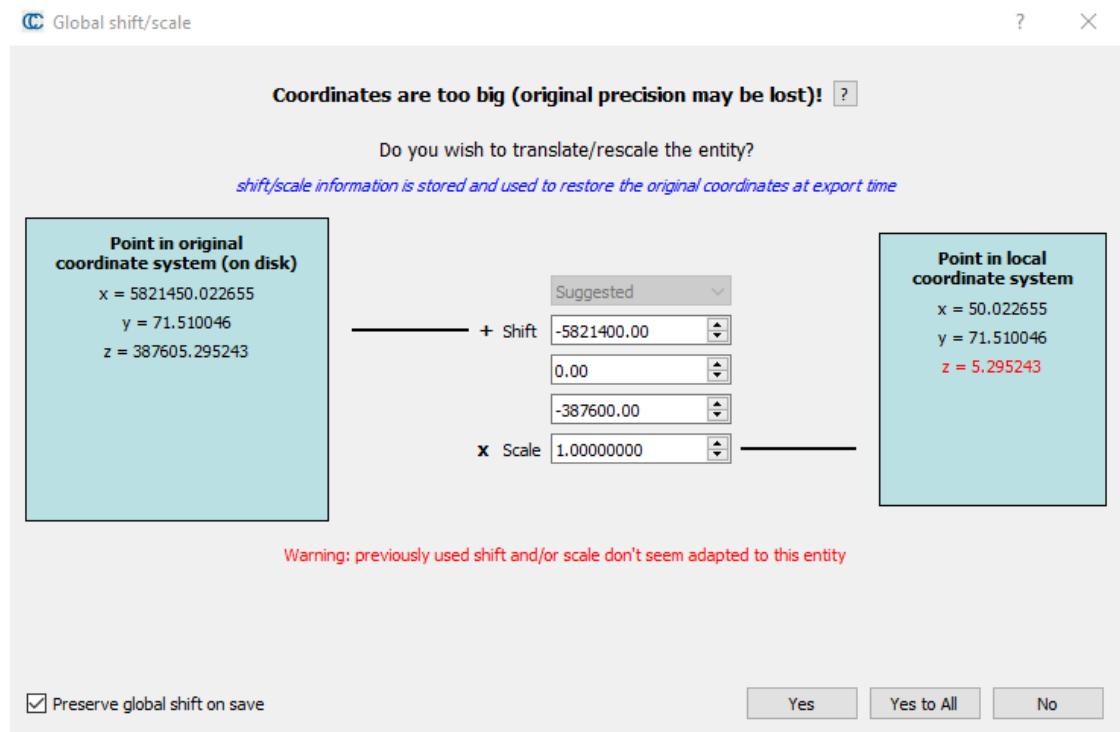


Abbildung 8: Dialog für Verschiebung des Koordinatensystems beim Import der UTM-Punktwolke in CloudCompare

Die exportierten Punkt wolken (.xyz Dateien) können mit dem Open-Source-Programm *CloudCompare* [51] geöffnet werden. Werden die Punkt wolken in einem anderen, gängigeren Format benötigt (zum Beispiel .ply oder .las/.laz), so können diese Formate aus *CloudCompare* exportiert werden.

Beim Import der *utm.xyz* Datei in *CloudCompare* wird der Nutzer zunächst nach dem Format der Attribute gefragt. Hierbei müssen die in Tabelle 1 beschriebenen Felder richtig zugeordnet werden (Positionen als *coord. X/Y/Z*, Farben als Farbwerte 0-255 und alle anderen Felder als *Scalar*). In den meisten Fällen ist die Zuordnung hier automatisch korrekt, sie solle aber dennoch überprüft werden, da manchmal eine Gruppe von Skalarfeldern von *CloudCompare* als Koordinaten des Normalenvektors interpretiert werden. Schließlich sollte vor Beginn des Imports

geprüft werden, ob ein Komma als Separator konfiguriert ist. Analog sollte beim Import der *raw.xyz*-Datei vorgegangen werden, falls diese analysiert bzw. visualisiert werden soll.

Nach dem Start des Imports der UTM-Punktwolke wird *CloudCompare* eine Transformation für die Koordinaten vorschlagen<sup>8</sup> (siehe Abbildung 8). Dies dient dazu, die Koordinatenwerte für die Darstellung zu verkleinern, was die Fließkommapräzision, im Vergleich zu sehr großen Koordinaten, verbessert. Diesem Prozess kann ohne Probleme zugestimmt werden.



(a) Ungefilterte Punktwolke aus einem Scan. Es sind viele Ausreißer in der Mitte des Scans erkennbar.



(b) Dieselbe Punktwolke aus Abbildung 9a, aber nach Entfernung der „schlechteren“ Hälften der Punkte nach Positionsgenauigkeit und Confidence

Abbildung 9: Filteroption in CloudCompare visualisiert anhand eines Scans der App

Die mit der App erstellten Punktwolken können nun in *CloudCompare* visualisiert werden, entweder mit RGB-Werten, oder nach Skalarfeldern wie der Distanz zum Smartphone, der Confidence, Genauigkeitswerten der Geospatial API, oder der Scan-ID. Aber auch verschiedene Nachbearbeitung, zum Beispiel Filter nach diesen Skalarfeldern (siehe Abbildung 9), manuelles Zuschneiden der Punktwolken, oder sogar die Erstellung von Meshes und Estimation von Oberflächen, sind mit der Software möglich.

Zur Analyse der Kamerapfade können sowohl *CloudCompare*, als auch *Google My Maps* ein-

---

<sup>8</sup>Bei besonders kleinen UTM-Koordinaten wahrscheinlich nicht, aber solche Koordinaten sind selten

gesetzt werden. Die Datei *frames\_utm.csv* kann in *CloudCompare* importiert werden, unter Beachtung der richtigen Zuordnung der Felder nach Tabelle 1. So sind die Kamerapositionen in *CloudCompare* verfügbar. Sie können jedoch auch auf einer Karte in *Google My Maps* angezeigt werden. Hierfür kann die *frames.kml* Datei direkt als Ebene in *My Maps* importiert werden. Die Kamerapfade werden dann nach ihrer Scan-ID farbig gekennzeichnet. Sie werden als Linie und nicht als Sammlung von Punkten importiert, da *My Maps* maximal 2000 „Features“ pro Ebene unterstützt, und Linien als ein Feature gelten, während bei Punkten jeder Punkt ein Feature ist.

## 4.2 Funktionsweise des Scanvorgangs

### 4.2.1 Prozesse pro Frame

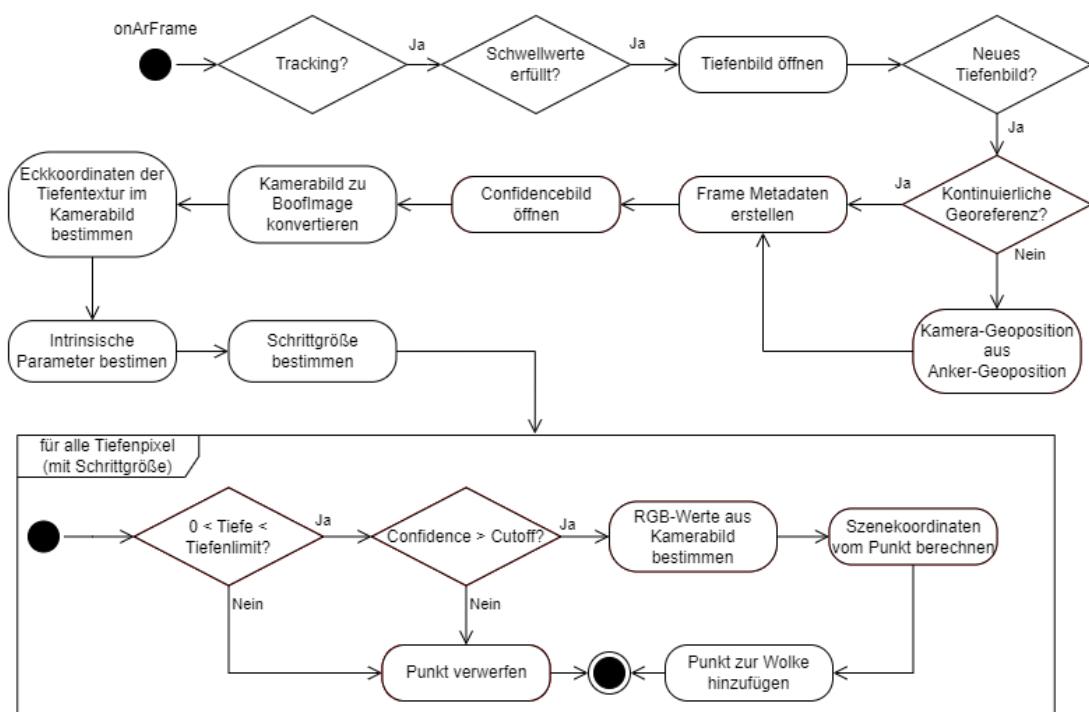


Abbildung 10: Schematische Darstellung der wichtigsten Schritte im Scanprozess bei der Verarbeitung von Frames

Wenn der Scanvorgang, also die Aufzeichnung von Punkten, beginnt, dann muss jeder neue Frame des AR-Bildes verarbeitet werden. Eine schematische Abbildung dieses Ablaufes ist zur Übersicht in Abbildung 10 dargestellt.

Die Verarbeitung des Frames beginnt damit, dass evaluiert wird, ob ARCore gerade die Position korrekt trackt. Dies ist zum Beispiel nicht der Fall, wenn das Gerät zu schnell bewegt wurde, oder die Kamera temporär verdeckt war. Ist ARCore im TRACKING-Status, kann die Bearbeitung

des Frames fortgesetzt werden. Im gleichen Zug wird ebenfalls geprüft, ob die für den Scan konfigurierten Schwellwerte für die Genauigkeit der Kameraposition nicht überschritten sind. So werden Frames direkt ausgeschlossen, wenn die Kameraposition zu ungenau ist.

Nun kann das Tiefenbild aus ARCore geladen werden. Hierbei wird „Raw Depth“ von ARCore genutzt, da dort genauere Tiefendaten als beim regulären Tiefenbild vorliegen. Dafür ist für nicht alle Pixel im Bild ein Tiefenwert vorhanden. Das Tiefenbild besitzt einen Zeitstempel, welcher gegen den des zuletzt genutzten Tiefenbilds verglichen werden kann. Ist der Zeitstempel gleich, so wurde für den aktuellen Frame kein neues Tiefenbild erstellt. Stattdessen handelt es sich um eine reprojizierte Version des alten Tiefenbilds, weshalb keine neuen Daten vorliegen und das Tiefenbild für diesen Frame verworfen werden kann. Ist es hingegen ein neuer Zeitstempel, handelt es sich um ein neues Tiefenbild und die Verarbeitung des Bildes kann fortgesetzt werden.

Nun können die Metadaten für den Frame erstellt werden, da das Tiefenbild für diesen Frame ausgewertet wird. Hierfür ist die aktuelle Geoposition des Geräts erforderlich. Bei kontinuierlicher Georeferenz ist diese direkt gegeben, doch bei nicht-kontinuierlicher Georeferenz muss die Kamera-Geoposition erst mit der Geoposition des Ankers und der relativen Position der Kamera zum Anker berechnet werden. Hierfür wird die Methode aus Kapitel 2.5 angewandt.

Im Anschluss werden die Buffer für das Auslesen der Tiefen- und Confidencebilder erstellt, da die Buffer der jeweiligen Bilder nicht direkt gelesen werden können, sondern erst kopiert werden müssen. Hierbei muss beachtet werden, dass die `ByteBuffer` beider Bilder die Byte-Reihenfolge *Little Endian* nutzen, entgegen dem für Java (und damit Kotlin) üblichen *Big Endian*.

Als Nächstes wird das aktuelle CPU-Kamerabild aus ARCore in ein BoofCV-Bild (`BoofImage`) im `RGB`-Format umgewandelt. Das CPU-Kamerabild ist niedriger aufgelöst als die GPU-Textur, welche dem Nutzer auf dem Bildschirm angezeigt wird, kann dafür aber direkt pro Frame durch einen einfachen Methodenaufruf abgerufen werden, weshalb es hier als Kamerabild für die Farben ausgewählt wurde. Das Framework BoofCV [52] wird hier verwendet, da es effiziente Java-Methoden zur Verfügung stellt, um die von ARCore gelieferten Bilder vom `YUV-420`-Format in `RGB` umzuwandeln. Aus dem `RGB`-Bild können dann einfach die Farbwerte nach Bedarf gelesen werden.

Die Auflösung (und das Seitenverhältnis) von Tiefenbild und dem CPU-Kamerabild stimmen standardmäßig nicht überein, da das Tiefenbild das gleiche Seitenverhältnis (aber geringere Auflösung) wie die GPU-Textur von ARCore hat, während das CPU-Kamerabild ein anderes Seitenverhältnis und eine andere Auflösung besitzt. Zur Umwandlung kann die Methode `Frame.transformCoordinates2d()` genutzt werden [53]. In frühen Versionen der App wurde diese Umwandlung noch pro Pixel vollzogen, aber es wurde festgestellt, dass die Methode so rechenaufwendig ist, dass die Bearbeitungszeit pro Frame signifikant verlängert wurde (siehe auch Kapitel 5.6). Um dem entgegenzuwirken, wird die Methode nun vor der Bearbeitung aller Pixel für nur die Eckpunkte (oben-links und unten-rechts) des Bildes durchgeführt. Damit sind die transformierten Minima und Maxima von jeweils der  $x$ - und  $y$ -Achse bekannt. So können später normalisierte Koordinaten genutzt werden, um zwischen den bekannten Grenzen zu interpolieren.

Nach der Berechnung der Eckkoordinaten im Kamerabild werden die intrinsischen Parameter des Tiefenbilds bestimmt. Dabei wird die in Kapitel 2.4 beschriebene Methode angewandt, um die intrinsischen Parameter der GPU-Textur auf das Tiefenbild zu skalieren.

Schließlich wird die Schrittgröße  $step$  zum Iterieren der Tiefenpixel berechnet. Hierfür wird die Quadratwurzel aus dem Verhältnis zwischen Pixelanzahl im Tiefenbild und der in der Scankonfiguration festgelegten maximalen Punkteanzahl pro Frame ( $p_{max}$ ) gebildet und auf die nächste ganze Zahl aufgerundet:

$$v = \sqrt{\frac{\text{Breite} \cdot \text{Höhe}}{p_{max}}}$$

$$step = \lceil v \rceil$$

Diese Technik zur Begrenzung der Menge an bearbeiteten Pixeln wurde aus [29] übernommen und sorgt dafür, dass wenn weniger Tiefenpixel als  $p_{max}$  vorhanden sind, der  $step$  auf 1 gesetzt wird und so alle Pixel verarbeitet werden.

Somit ist nun die Bearbeitung der einzelnen Pixel vorbereitet. Für jeden Pixel kann der Tiefenwert, der Confidence-Wert, sowie der RGB-Farbwert bestimmt werden. Die intrinsischen Parameter des Tiefenbilds sind ebenfalls zur Berechnung der realen Position der Pixel vorhanden.

#### 4.2.2 Prozesse pro Pixel

Die Pixel können nun einzeln bearbeitet werden, mit zwei geschachtelten Schleifen, welche die im obigen Kapitel bestimmte Schrittzahl  $step$  nutzen. In Abbildung 10 ist diese Schleife schematisch in der unteren Box abgebildet. Im ersten Schritt wird validiert, ob überhaupt ein Tiefenwert für den gegebenen Pixel vorliegt. Dies ist der Fall, wenn eine Tiefe  $> 0m$  vorliegt und ein Confidence-Wert  $> 0$ . Außerdem wird validiert, dass die Tiefe nicht größer als das für den Scan konfigurierte Tiefenlimit ist, und dass der Confidence-Wert nicht unter dem konfigurierten Schwellwert liegt. Sind diese Voraussetzungen nicht gegeben, so wird der Pixel übersprungen. Sonst kann mit der Bestimmung der RGB-Werte fortgefahrene werden.

Zur Bestimmung des RGB-Werts für den Pixel werden aus dem oben erstellten `BoofImage` die RGB-Werte ausgelesen. Die äquivalenten Koordinaten werden dabei mithilfe der im Voraus bestimmten Minima und Maxima der jeweiligen Achsen ( $x_{min}$ ,  $x_{max}$ ,  $y_{min}$  und  $y_{max}$ ) und normalisierten Koordinaten berechnet:

$$x_{normalisiert} = \frac{x_{Tiefenbild}}{\text{Breite}_{Tiefenbild}}$$

$$y_{normalisiert} = \frac{y_{Tiefenbild}}{\text{Höhe}_{Tiefenbild}}$$

$$x_{BoofImage} = x_{min} + ((x_{max} - x_{min})) \cdot x_{normalisiert}$$

$$y_{BoofImage} = y_{min} + ((y_{max} - y_{min})) \cdot y_{normalisiert}$$

Zuletzt wird die Position des Pixels relativ zur Position des Ankers mithilfe der intrinsischen Parameter nach dem in Kapitel 2.4 vorgestellten Verfahren bestimmt und der Punkt wird zur partiellen Punktwolke des Frames hinzugefügt. Nachdem alle Pixel eines Frames bearbeitet wurden, werden alle gefundenen Punkte, sowie die Frame-Metadaten,persistiert und der nächste Frame kann bearbeitet werden.

### 4.3 Funktionsweise des UTM Postprocessors

Der UTM Postprocessor dient zur Georeferenzierung der Punktwolke im UTM Koordinatensystem nach Abschluss des Scavorgangs. In diesem Kapitel soll die Funktionsweise genauer beschrieben werden, für die Nutzung kann auf Kapitel 4.1.6 verwiesen werden.

Zur Umwandlung von Längen- und Breitengraden zu UTM-Koordinaten wird die Bibliothek *Coordinate Transformation Suite* (kurz *CTS*) [54] genutzt. Diese bietet zusätzlich die Funktionalität, den EPSG-Code der für eine bestimmte Position zuständigen UTM-Zone zu ermitteln.

Zu Beginn wird vom UTM Postprocessor die zu verwendende UTM Zone bestimmt. Diese wird entweder per EPSG-Code direkt aus der Scankonfiguration geladen (weil schon eine UTM-Zone für die Konfiguration festgelegt ist) oder durch die Konfiguration des UTM Postprocessors überschrieben (vgl. Kapitel 4.1.6). Ist der EPSG-Code der Scankonfiguration oder der überschriebene EPSG-Code der Postprocessor-Konfiguration leer, so wird die erste Kameraposition des Scans genutzt, um automatisch die zuständige UTM-Zone zu identifizieren.

Im Anschluss werden die Kamerapositionen in Längen- und Breitengraden nacheinander aus der Datei *frames.csv* (Frame-Metadaten) geladen, mit der Bibliothek in UTM-Koordinaten gewandelt, und schließlich in die Datei *frames\_utm.csv* geschrieben.

Nun wird die Geoposition der Punkte bestimmt. Hierfür wird jeder Punkt nacheinander aus der Datei *raw.xyz* geladen. Zusätzlich werden die zum Punkt gehörigen Frame-Metadaten mit der Kameraposition benötigt. Diese werden anhand der Frame-ID geladen. Somit ist die Position der Kamera in Geokoordinaten, die Position der Kamera relativ zum Anker des Scans, sowie die Position des jeweiligen Punktes relativ zum Anker des Scans bekannt. Damit kann das in Kapitel 2.5 vorgestellte Verfahren genutzt werden, um die Geokoordinaten des Punkts zu bestimmen. Dieser Vorgang wird dann für alle Punkte wiederholt und die berechneten Punkte mit UTM-Koordinaten werden in die Datei *utm.xyz* geschrieben. Sofern konfiguriert, wird auch ein Teil der Frame-Metadaten, spezifisch die Genauigkeitsangaben und die Scan-ID, zu den Punkten hinzugefügt. Dies ermöglicht die direkte Analyse der Punktwolken mit diesen Daten, ohne dass manuell die dazugehörigen Frame-Metadaten gelesen werden müssen.

## 4.4 Architektur

### 4.4.1 Erweiterbare Postprocessor-Architektur

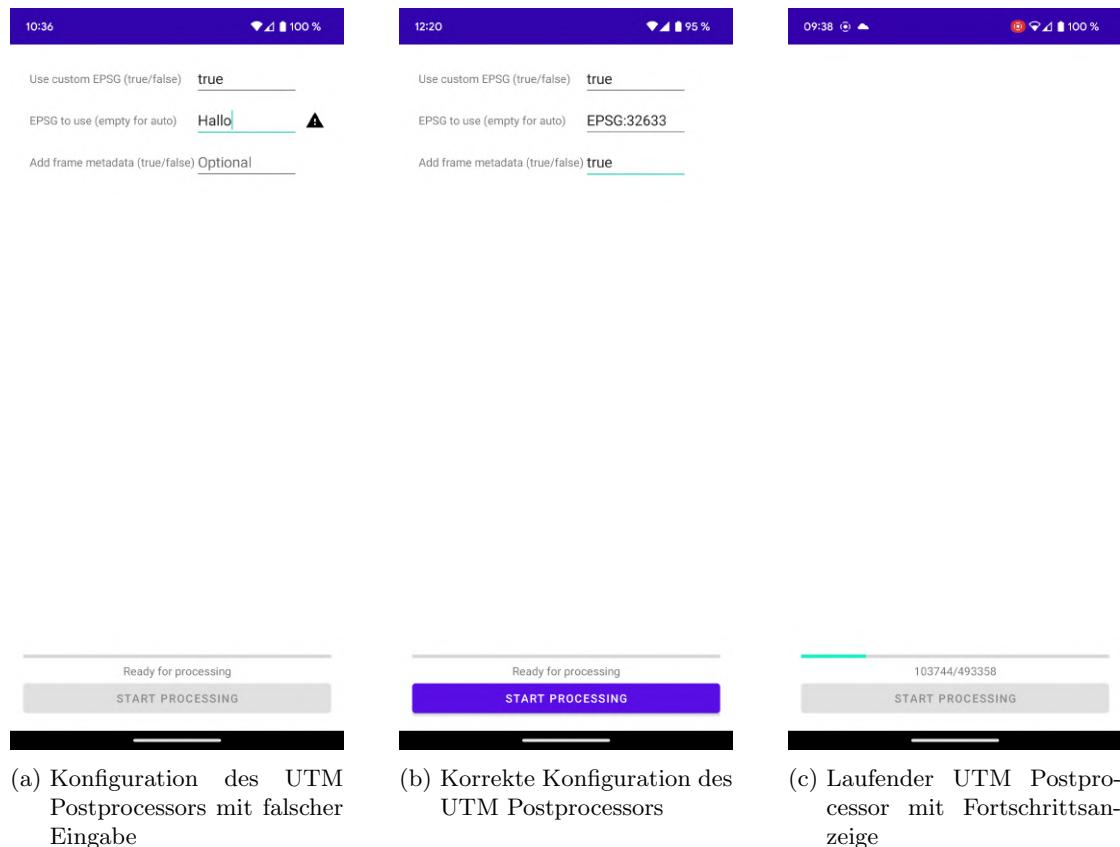


Abbildung 11: Konfiguration und Ausführung des UTM Postprocessors

Die App enthält eine dynamische Postprocessor-Architektur, damit neue Postprocessor zur Nachbearbeitung von Punktwolken einfach implementiert werden können. Die App selbst implementiert zwei Postprocessor, einen zur Georeferenzierung der Punktwolken aus den Geopositionen der Kamera (UTM Postprocessor), und einen zum Erstellen einer KML-Datei, welche die Kamerapfade eines Scans enthält.

Grundsätzlich werden alle verfügbaren Postprocessor in der Detailansicht der Scankonfiguration als Liste unter der „Scan“-Schaltfläche angezeigt (vgl. Kapitel 4.1.4). Sie sind nur dann auswählbar, wenn der Postprocessor auf diese Konfiguration angewandt werden kann (z.B. nur wenn Punkte vorhanden sind oder die Scankonfiguration georeferenziert ist), sonst ist sie ausgrau. Nachdem ein Postprocessor ausgewählt wurde, kann der Vorgang konfiguriert werden (siehe Abbildung 11a). Die Eingaben in die Felder werden dabei direkt bei der Eingabe validiert und bei falschen oder fehlenden Eingaben wird ein entsprechendes Warnsymbol neben dem

Textfeld angezeigt und die Schaltfläche zum Starten wird deaktiviert (auch in Abbildung 11a zu sehen). Die Konfigurationsmöglichkeiten werden dabei nur in Code definiert, die Anzeige wird dann automatisch erweitert.

Ist die Konfiguration korrekt, kann der Postprocessor durch Auswahl der „Start Processing“-Schaltfläche gestartet werden (siehe Abbildung 11b). Während der Ausführung wird eine Fortschrittsanzeige für den Postprocessor angezeigt (siehe Abbildung 11c).

Um einen neuen Postprocessor zu erstellen, muss zunächst das `PostProcessor` Interface implementiert werden. Dieses definiert, wie der Postprocessor konfiguriert wird (`getConfig()`- und `configure()`-Methoden), und was der Postprocessor tut (`process()`-Methode).

Zur Konfiguration eines Postprocessors geben Nutzer in eine Liste von Konfigurationen Werte ein (siehe Abbildung 11a). Der Postprocessor muss definieren, welche Felder vorhanden sind, ob sie erforderlich oder optional sind, und welche Werte akzeptiert werden. Hierfür muss für jedes Konfigurationsfeld das Interface `PostProcessingConfig` implementiert werden. Hier kann dann festgelegt werden, ob das Konfigurationsfeld erforderlich oder optional ist (`required`-Feld), wie die Konfiguration heißt (via String-Resource-ID, `name`-Feld), und wie die Freitexteingabe der Konfiguration validiert wird (`validateValue()`-Methode). In der `getConfig()`-Methode des Postprocessors muss dann eine Liste dieser Objekte zurückgegeben werden. Diese Liste kann natürlich auch leer sein, wenn keine Konfiguration erforderlich ist. Bevor der Postprocessor gestartet wird, wird schließlich die Methode `configure()` aufgerufen, welche eine Map zur Zuordnung von Strings zu `PostProcessingConfigs` enthält. Hier sollten dann im `PostProcessor`-Objekt Variablen anhand der konfigurierten Werte initialisiert werden, damit die Konfiguration für die Bearbeitung zur Verfügung steht. Wenn `configure()` aufgerufen wird, sind alle Eingaben schon mit den Konfigurationsobjekten validiert worden, sodass keine weitere Validierung erforderlich ist.

Schließlich muss die `process()`-Methode des `PostProcessor`-Interface implementiert werden. Hier wird die eigentliche Verarbeitung des Scans durchgeführt. Die Methode wird als Kotlin Coroutine [55] asynchron ausgeführt und kann seinen Fortschritt mithilfe von Kotlin Flow [56] durch `Progress`-Objekte zurückliefern. Diese Objekte enthalten einen ganzzahligen Fortschritt (0-100), mit welchem die Fortschrittsanzeige angepasst wird, und einen Informationstext, welcher über der Fortschrittsanzeige dargestellt wird (siehe Abbildung 11c). Es muss darauf geachtet werden, dass die Coroutine nicht auf dem Hauptthread ausgeführt wird (z.B. möglich mit dem `flowOn(Dispatchers.IO)`-Operator). Sobald als Fortschrittwert 100 zurückgeliefert wird, wird der `PostProcessor` beendet.

In der `process()`-Methode empfiehlt es sich die `processFrameMetadata` und `processRawData` Hilfsmethoden der `ScanRepository`-Klasse zu nutzen. Diese lesen automatisch die Frame-Metadaten oder die Punktwolke der `raw.xyz`-Datei Zeile für Zeile und es kann eine Funktion zur Bearbeitung der Zeilen übergeben werden. Der Rückgabewert dieser Funktion (ein `String`) wird dann als neue Zeile in die angegebene, neue Datei geschrieben. Für eine beispielhafte Implementierung kann der UTM Postprocessor betrachtet werden. Bei Bedarf kann natürlich auch

auf die Dateien manuell zugegriffen werden.

Ist der `PostProcessor` fertig implementiert, muss er nur noch registriert werden, damit er auch in der Detailansicht der Scankonfiguration angezeigt wird. Hierfür muss nur die abstrakte Klasse `PostProcessorInfo` implementiert werden, wo ein eindeutiger String zur Identifikation (`identifier`-Feld), der Name des Postprocessors in der Detailansicht der Scankonfiguration (`name`-Feld) und die Erstellung des jeweiligen `PostProcessor` Objekts (`factory`-Feld) festgelegt wird. Außerdem kann die Funktion `isApplicable()` überschrieben werden, um die Scankonfiguration zu prüfen und die Nutzung des Postprocessors zu erlauben oder zu verbieten (die Schaltfläche für den Postprocessor wird dann ausgegraut). Nach Implementation der `PostProcessorInfo`-Klasse ist der neue Postprocessor registriert, da die Implementationsen der Klasse programmatisch von der App gesucht werden.

#### 4.4.2 Wichtige Klassen

Im Folgenden werden die Funktionen einiger wichtiger Klassen beziehungsweise Dateien dokumentiert. Dies soll dazu dienen, den Code eines Prozesses schnell im Projekt zu finden und einen Überblick über die Architektur zu erhalten.

**ArScanFragment** Diese Klasse verwaltet die Anzeige und Prozesse im Scavorgang und initialisiert ARCore.

**FrameProcessor** In dieser Klasse werden die einzelnen Frames verarbeitet. Hier werden die Bedingungen zur Auswertung von Frames geprüft und die Tiefenbilder, sowie das CPU-Bild geladen.

**FrameMetaData** Diese Klasse verwaltet die Metainformationen eines Frames, also die Frame-ID, die Scan-ID, die Kameraposition relativ zum Anker, und bei Bedarf die Geoposition der Kamera, mit den Genauigkeitsangaben der Geospatial API.

**FrameImageData** Mit dieser Klasse werden Tiefenbild, Confidence-Bild und das CPU-Bild des Frames verwaltet. Hier können Objekte des Typs `PixelData` für gegebene Koordinaten aus den gegebenen Bildern erstellt werden.

**PixelData** Diese Datenklasse enthält alle berechneten Daten eines Tiefenpixels, nämlich die Position, Confidence, Farbwerte und die Frame-ID.

**DepthCameraIntrinsics** Diese Klasse ermöglicht die Berechnung von intrinsischen Parametern des Tiefenbilds und von Szenekoordinaten eines gegebenen Tiefenpixels.

**FramePointCloud** Objekte dieser Klasse enthalten `FrameMetaData` und eine Reihe von `PixelData`-Objekten. Sie repräsentieren die Punktwolke eines Frames und werden nach Bearbeitung eines Framespersistiert.

**ScanRepository** In dieser Klasse ist der Zugriff auf die Dateien zentralisiert. Sie ist für das Laden und Speichern der Punktwolken und Scankonfigurationen zuständig.

**TimingHelper** Mit dieser Klasse kann die Dauer bestimmter Prozesse gemessen und ausgegeben werden. Alle gemessenen Prozesse bekommen einen Namen (String) zur Identifikation und nach der Bearbeitung eines Frames werden alle gemessenen Zeiten auf dem Bildschirm ausgegeben.

**GeoPoseHelper** Diese Datei enthält mehrere *Kotlin Extension Functions* zur Umwandlung globaler Koordinaten und zur Berechnung von Koordinaten anhand kartesischer Offsets (vgl. Kapitel 2.5).

**UtmCoordinateConverter** Mit dieser Klasse können Koordinaten mit Längen- und Breitengrad in UTM-Koordinaten umgewandelt werden.

## 5 Untersuchungen

In Bezug auf die Zielsetzung der Arbeit, „möglichst genaue“ Punktwolken in Straßenzügen zu erstellen, soll mit den Untersuchungen ermittelt werden, welche Genauigkeiten bei der Erhebung von Punktwolken mit der App zu erwarten sind. Dabei sind vier Aspekte besonders relevant und sollen bestimmt werden:

- Genauigkeit (Streuung und Fehler) der Depth API
- Genauigkeit der Positionsbestimmung und der Genauigkeitsangaben der Geospatial API
- „Passform“ von unterschiedlichen Scans an der gleichen Position, also wie gut passen unterschiedliche Scans an den Grenzen zusammen (relevant, da großflächige Scans von Straßenzügen in Segmenten erstellt werden müssen)
- Fehler in der Position der Punktwolke bei einem weit entfernten, georeferenzierten Anker

### 5.1 Grundlagen zur Testdurchführung

#### 5.1.1 Untersuchung von Kamerapfaden

In ausgewählten Versuchen dieser Arbeit muss der Pfad der Kamera (bzw. des Smartphones) über den Testverlauf betrachtet werden. Hierfür wird der KML-Export (siehe Kapitel 4.1.7) der App genutzt. Die *frames.kml*-Datei wird als Ebene in *Google My Maps* [46] hochgeladen, um die Pfade der verschiedenen Scandurchläufe auf einer Karte zu visualisieren. Das auf der Webseite vorhandene Tool zur Distanzmessung kann genutzt werden, um Abweichungen zwischen den Kamerapfaden zu messen. In einigen Versuchen wird der Kamerapfad auch in *CloudCompare* analysiert. Hierfür kann die Datei *frames\_utm.csv* genutzt werden, welche vom UTM-Postprocessor (siehe Kapitel 4.1.6) erstellt wird.

#### 5.1.2 Mehrere Scans mit der gleichen Scankonfiguration

Viele der Versuche erfordern, dass mehrere Scans mit der gleichen Scankonfiguration durchgeführt werden. Hierbei wird der AR-Scan Bildschirm nach jedem Scan einmal verlassen. Dies erhöht die *Scan-ID*, welche in der resultierenden Punktwolke als Skalarfeld vorhanden ist (siehe Kapitel 4.1.8). Um die unterschiedlichen Scans anschließend in *CloudCompare* voneinander zu trennen, kann das *Filter points by value*-Werkzeug genutzt werden.

#### 5.1.3 Planarität und Oberflächenvarianz

In Versuch 1 werden zwei geometrische Eigenschaften von Punktwolken betrachtet. Zunächst die *Planarität*, welche die Tendenz einer Punktwolke zur Anordnung entlang einer Ebene beschreibt [57]. Planarität ist ein Wert zur (wahrscheinlichkeitsbasierten) Bewertung der „2-Dimensionalität“ einer Punktwolke, die jeweiligen 1D und 3D Gegenstücke sind *Linearität* und

*Streuung* [58]. Neben der Planarität wird die *Oberflächenvarianz* (engl. *surface variation*) betrachtet, die die Variation der Punkte entlang der Normalen der durch die Punkte erzeugten Ebene beschreibt. Diese kann genutzt werden, um Punktfolgen zu erkennen, in denen mehrere Oberflächen übereinander liegen [6, 59].

Zur Berechnung dieser Eigenschaften für einen spezifischen Punkt  $p$  muss zunächst die  $3 \times 3$  Kovarianzmatrix  $C$  bestimmt werden. Es gilt nach [58]:

$$C = \frac{1}{k+1} \sum_{i=0}^k (p_i - \bar{p}) \cdot (p_i - \bar{p})^T$$

wobei  $p_i$  die Nachbarn von  $p$  sind,  $k$  die Anzahl der nächsten Nachbarn von  $p$  beschreibt und  $\bar{p}$  der Centroid der Nachbarschaft von  $p$ . Die Nachbarschaft von  $p$  ist definiert aus allen Punkten innerhalb eines Radius  $r$  um  $p$ .

Aus der Kovarianzmatrix  $C$  können dann mit dem Eigenwertproblem die Eigenvektoren  $\vec{e}_0$ ,  $\vec{e}_1$  und  $\vec{e}_2$ , sowie die Eigenwerte  $\lambda_0$ ,  $\lambda_1$  und  $\lambda_2$  bestimmt werden [59]:

$$C \cdot \vec{e}_l = \lambda_l \cdot \vec{e}_l, l \in \{0, 1, 2\}$$

Die Eigenvektoren  $\vec{e}_l$  beschreiben ein orthogonales Koordinatensystem und die Eigenwerte  $\lambda_l$  beschreiben die Variation entlang der Richtungen der Eigenvektoren innerhalb der Nachbarschaft um den Punkt  $p$ . Aus den Eigenwerten können nun die gesuchten geometrischen Eigenschaften Planarität  $P$  und Oberflächenvarianz  $\sigma$  für den Punkt  $p$  bestimmt werden [57]:

$$P = \frac{\lambda_2 - \lambda_3}{\lambda_1}$$

$$\sigma = \frac{\lambda_3}{\lambda_1 + \lambda_2 + \lambda_3}$$

Die Werte für Planarität und Oberflächenvarianz sind komplett vom gewählten Nachbarschaftsradius abhängig. Unter Betrachtung einer Nachbarschaft von zum Beispiel 100m ist eine Punktfolge mit 1m Dicke planar, mit geringer Oberflächenvarianz. Bei einer Nachbarschaftsgröße von 1cm hingegen hat die gleiche Punktfolge keine Planarität und hohe Oberflächenvarianz. Normalerweise wird der zu verwendende Radius  $r$  für die Nachbarschaft aus der Dichte der Punktfolge errechnet [59]. Die in dieser Arbeit entwickelte App erstellt hingegen Punktfolgen mit variabler Dichte. Bleibt ein Ort länger im Bild, so sind für diesen Ort mehr Punkte vorhanden. War die Tiefenerkennung zu einem Zeitpunkt nicht möglich, so sind für Orte, die zu diesem Zeitpunkt gescannt wurden, weniger oder keine Punkte vorhanden. Somit ist die Dichte hierfür nicht nutzbar.

Es ist jedoch möglich, bei Betrachtung einer planaren Wand, den Nachbarschaftsradius iterativ zu erhöhen, bis sich die meisten Punkte eines Scans (also eine Spitze im Histogramm) im optimalen Bereich der jeweiligen Eigenschaft befinden (nahe 1 für Planarität und nahe 0 für

Oberflächenvarianz). Features mit einem Tiefenunterschied, der dem Radius der Nachbarschaft entspricht, sind dann in der Punktwolke als separate Ebenen zu erkennen und gehen nicht im Rauschen unter. So kann für eine gegebene Scandistanz der erforderliche Tiefenunterschied für klare Trennung von Ebenen ermittelt werden.

Die Berechnung von Planarität und Oberflächenvarianz kann mithilfe von *CloudCompare* automatisch erfolgen. Das Programm exportiert die jeweiligen Werte dann als Skalarfeld.

## 5.2 Versuch 1: Scanqualität im Verhältnis zum Abstand

**Zielsetzung:** Für diesen Versuch soll der Einfluss der Depth API auf die Scanqualität in Abhängigkeit vom Scanabstand ermittelt werden. Relevant ist hierbei der Tiefenfehler (werden Punkte zu nah oder zu weit weg erkannt), sowie die Streuung (wie „dick“ sind erkannte Oberflächen). Eine Folge aus der Streuung ist, wie groß der Tiefenunterschied von Features einer Wand sein muss, damit die nicht in der Streuung untergehen.

**Hypothese:** Es ist zu erwarten, dass der Tiefenfehler und die Streuung mit zunehmender Distanz zunehmen. Google gibt zwei Erwartungen für die Zuverlässigkeit der Depth API an unterschiedlichen Stellen an. Zum einen gibt es angeblich „[...] die genauesten Ergebnisse [...] wenn [sich] das Gerät einen halben bis fünf Meter von der realen Szene [...]“ [18] (übersetzt aus dem Englischen) entfernt befindet. An anderer Stelle heißt es hingegen „Optimale Tiefengenauigkeit tritt zwischen 50cm und 15m [Entfernung] von der Kamera auf [...]“ [23] (übersetzt aus dem Englischen). Da die Depth API für Okklusion und nicht zur Punktwolkenerhebung, und damit mit geringeren Genauigkeitsanforderungen entwickelt wurde [1], sind relativ hohe Werte für Streuung und Fehler, insbesondere bei größerer Distanz, zu erwarten. Unterhalb der von Google angegebenen 15m werden weniger als 1m Fehler und Streuung erwartet.

### 5.2.1 Durchführung

Zur Bewertung der Scanqualität, insbesondere der Streuung, kann eine gerade Wand betrachtet werden. Wie gerade eine Wand aus Punkten ist, lässt sich mithilfe von Planarität und Oberflächenvarianz bestimmen. Bei viel Streuung der Wand sind auch schlechtere Werte für Planarität und Oberflächenvarianz zu erwarten. Außerdem kann die Abweichung von der tatsächlichen Wand, sowie die Breite der Streuung der Punkte bestimmt werden, wenn der tatsächliche Abstand zur Wand von der Messposition mit dem Abstand zur erkannten Wand verglichen wird.

Für den Versuch wurde eine gerade Hauswand mit leichter Textur ausgewählt (siehe Abbildung 12). Der Abstand zur Hauswand wurde mit einem Laser-Distanzmesser (Modell: *Tacklife S2 80m*) gemessen, sodass die wahre Distanz bekannt ist. Es wurde in den Abständen 1m, 2m, 3m, 6m, 8m, 10m und 16m von der Wand ein Scan durchgeführt. Diese Abstände, insbesondere die Lücke zwischen 3m und 6m waren durch die realen Bedingungen vor Ort (parkende Autos) bedingt. Vor Beginn eines Scans wurde das Smartphone einen Meter parallel zur Hauswand bewegt, damit die Depth API die Tiefe bestimmen kann, bevor der Scan beginnt und so



Abbildung 12: Die untersuchte Hauswand. Der untersuchte Wandbereich wurde rot markiert und ist ca. 90cm breit, 2,20m hoch und steht 10cm vom Rest der Wand hervor.

initiale Fehlerkennungen nicht Teil des Scans sind (vgl. Kapitel 5.6). Bei jedem Scan wurde das Smartphone dann schließlich um 2 Meter parallel zur Wand bewegt (siehe auch Abbildung 13). Alle Scans wurden mit der gleichen Scankonfiguration durchgeführt, mit einem Depth-Limit von 40m und keiner kontinuierlichen Georeferenz. Die anderen Parameter wurden nicht verändert und hatten ihren Standardwert (siehe Kapitel 4.1.3).

Nach dem Scanvorgang wurde für die verwendete Scankonfiguration der UTM-Postprocessor ausgeführt. Während die globale Position in diesem Test nicht bewertet wird, ist es dennoch vorteilhaft, alle Scans am ungefähr gleichen Ort in der Software zu haben. Die UTM-Punktwolke, sowie die UTM-Kamerapositionen wurden im Anschluss in die Software *CloudCompare* importiert.

Die Punktwolke wurde nun mit dem *Cross Section*-Werkzeug auf den untersuchten Wandbereich zugeschnitten. Hierbei wurde mit den Farbwerten verifiziert, dass der korrekte Bereich gewählt wurde. Schließlich wurden die Scans aus unterschiedlichen

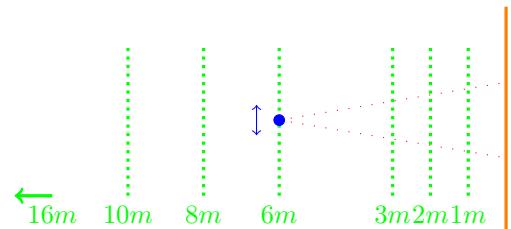


Abbildung 13: Wand (orange) und Kamerapfade (grün) für die Scans aus unterschiedlichen Entfernungen (Aufsicht). Eine Kameraposition ist beispielhaft mit der Bewegungsrichtung und dem Sichtkegel gekennzeichnet.

Entferungen anhand ihrer Scan-ID mithilfe des *Filter points by value*-Werkzeugs getrennt (vgl. Kapitel 5.1.2).

Zur Untersuchung von Planarität und Oberflächenvarianz wurde das Werkzeug *Compute geometric features* aus *CloudCompare* auf alle Scans angewandt. Dabei wurde zunächst eine konstante Nachbarschaftsgröße von  $0,1m$  für alle Scans genutzt, welche dann iterativ erhöht wurde (in  $0,1m$  Schritten), um die erkennbaren Tiefenunterschiede für eine gegebene Distanz zu ermitteln (siehe Kapitel 5.1.3).

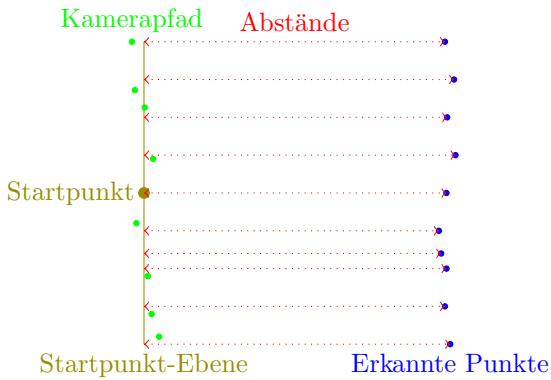


Abbildung 14: Auswertung des Abstands (rot) zwischen erkannten Wandpunkten (blau) und der Ebene des Startpunkts des Kamerapfads (oliv). Die Punkte des Kamerapfads sind grün gekennzeichnet. (Aufsicht)

Zur Untersuchung der Genauigkeit der gescannten Distanz musste der Abstand zwischen der Kameraposition beim Start des Scans (welche mit dem Laserscanner verifiziert wurde) und den erkannten Punkten bestimmt werden. Das Skalarfeld zur Kameradistanz, welches auch in den Punktwolken vorhanden ist, konnte nicht genutzt werden, da zwar die initiale Position der Kamera mit dem Laser-Distanzmessgerät verifiziert wurde, jedoch nicht der gesamte Kamerapfad. Dieser hat leicht ( $\pm 10cm$ ) variiert, da er nicht perfekt abgelaufen werden konnte. Zudem wurden durch das Sichtfeld diese Distanzen auch nicht immer orthogonal gemessen, für die Analyse wird jedoch die orthogonale Distanz benötigt. Zur Distanzbestimmung zwischen initialer Kamera position und den erkannten Punkten mussten zunächst die UTM-Kamerapositionen importiert werden. Dann wurde die Orientierung der Wandebene mithilfe des Scans aus  $2m$  Entfernung und dem *Fit Plane*-Werkzeug näherungsweise bestimmt. Dann wurden Ebenen mit demselben Normalenvektor, aber an den jeweils ersten Positionen der Scans erstellt, sodass für jeden Scan eine Ebene an der ersten Kameraposition des Scans parallel zur Ebene der Wand existiert. Dies ist in Abbildung 14 schematisch für einen Scanabstand dargestellt.

Schließlich wurden dann mit dem *Cloud/Primitive distance*-Werkzeug die Abstände der jeweiligen Punkte der Punktwolken zu den jeweiligen Ebenen bestimmt. Die resultierenden Werte wurden als CSV exportiert und zu einem Boxplot-Diagramm zur Auswertung zusammengefasst.

### 5.2.2 Ergebnisse

**Planarität und Oberflächenvarianz** Bei der Untersuchung der Oberfläche wurden Planarität und Oberflächenvarianz zunächst bei einer Nachbarschaftsgröße von  $10\text{cm}$  berechnet. Die Histogramme für diese Werte sind in den Abbildungen 15 und 16 zu sehen. Es gilt zu beachten, dass bei einem perfekten Scan der Wand eine Planarität von 1 und eine Oberflächenvarianz von 0 für alle Punkte zu erwarten ist. Die Scans aus 1m und 2m Entfernung weisen noch sehr eindeutige Spitzen in diesen optimalen Bereichen für die Planarität und Oberflächenvarianz auf. Das heißt Features mit  $10\text{cm}$  (entspricht der Nachbarschaftsgröße) Tiefenunterschied sind bei diesen Scanabständen noch klar erkennbar. Bei einer Distanz von 3m hingegen beginnen die Werte sich breiter über das Spektrum zu verteilen. Zwar sind die Spitzen immer noch im bevorzugten Bereich der jeweiligen Eigenschaft, sodass Features einer Wand mit  $10\text{cm}$  Tiefenunterschied noch erkennbar sind, aber mehr Rauschen in den Daten sorgt dafür, dass die Features unklarer definiert sind. Ab einer Scandistanz von 6m lösen sich die Spitzen der Daten vom optimalen Bereich. Hier kann nun davon ausgegangen werden, dass Features mit  $10\text{cm}$  Höhenunterschied nicht mehr eindeutig definiert sind und im Rauschen der Oberfläche untergehen.

Nun kann man, wie in Kapitel 5.1.3 beschrieben, schrittweise die Nachbarschaftsgröße erhöhen, bis sich die Werte bei Scans aus einer Entfernung wieder an den optimalen Werten gruppieren, so wie es bei  $10\text{cm}$  Nachbarschaftsgröße bis 3m Entfernung der Fall ist. Die so bestimmten Werte sind in Tabelle 2 abgebildet.

Scanabstand	1m	2m	3m	6m	8m	10m	16m
Erkennbare Featuretiefe	0,05m	0,05m	0,10m	0,30m	0,30m	0,50m*	0,80m*

Tabelle 2: Scandistanzen und die dazugehörige Featuretiefe, die noch erkennbar ist. Bei Werten mit \* wurde nur die Oberflächenvarianz ausgewertet, da die untersuchte Oberfläche zu klein für Planarität mit so großer Nachbarschaft ist

Es fällt auf, dass die erforderliche Featuregröße mit dem Abstand steigt. Und bei einem Abstand von  $16\text{m}$  vom gescannten Objekt sind große Unterschiede in der Tiefe notwendig, damit Features überhaupt erkennbar sind. Strukturen wie Fensterrahmen oder Verzierungen der Fassade ( $\approx 10\text{cm}$  Tiefenunterschied) erfordern Scans aus nächster Nähe ( $< 3\text{m}$ ).

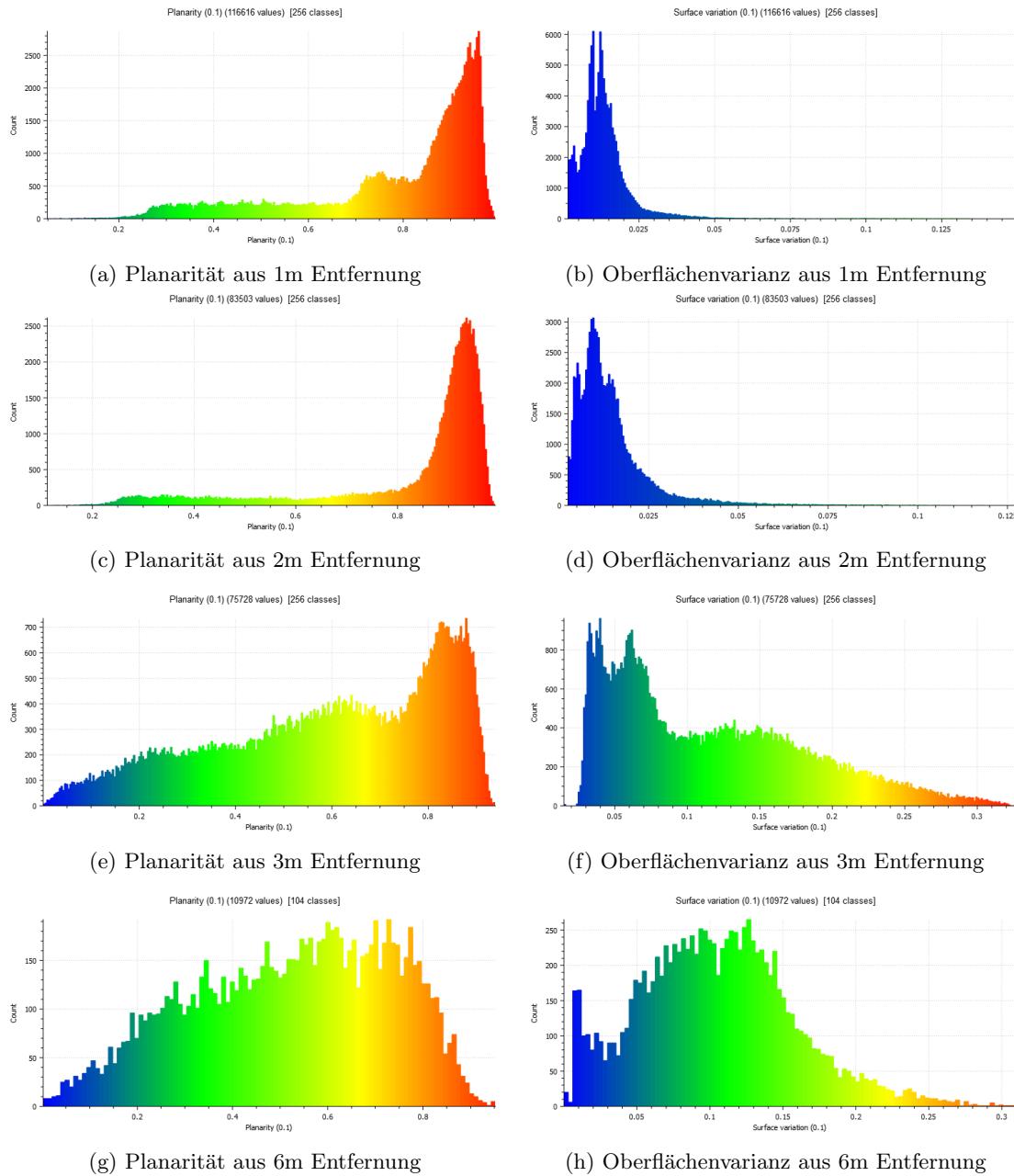


Abbildung 15: Histogramme der Planarität und Oberflächenvarianz für Scanabstände zwischen 1m und 6m

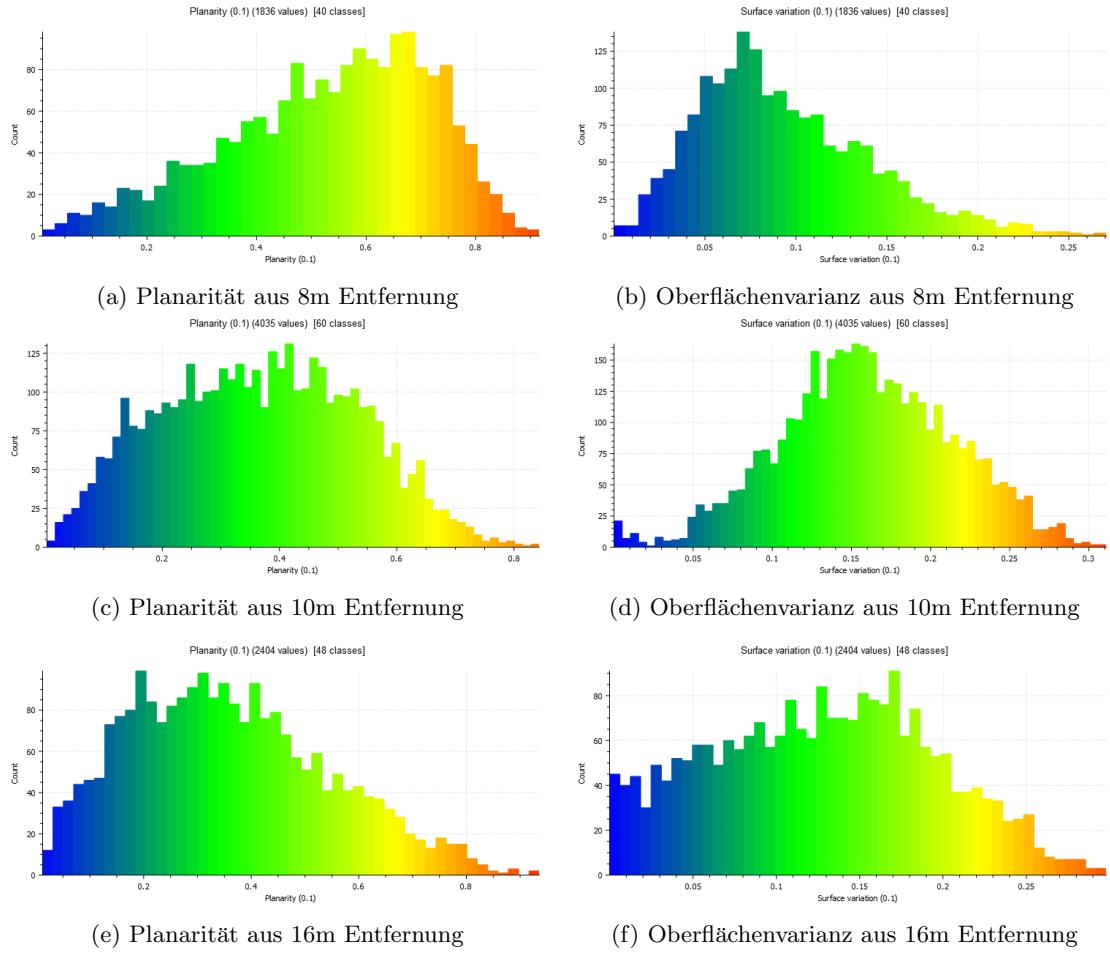


Abbildung 16: Histogramme der Planarität und Oberflächenvarianz für Scanabstände zwischen 8m und 16m

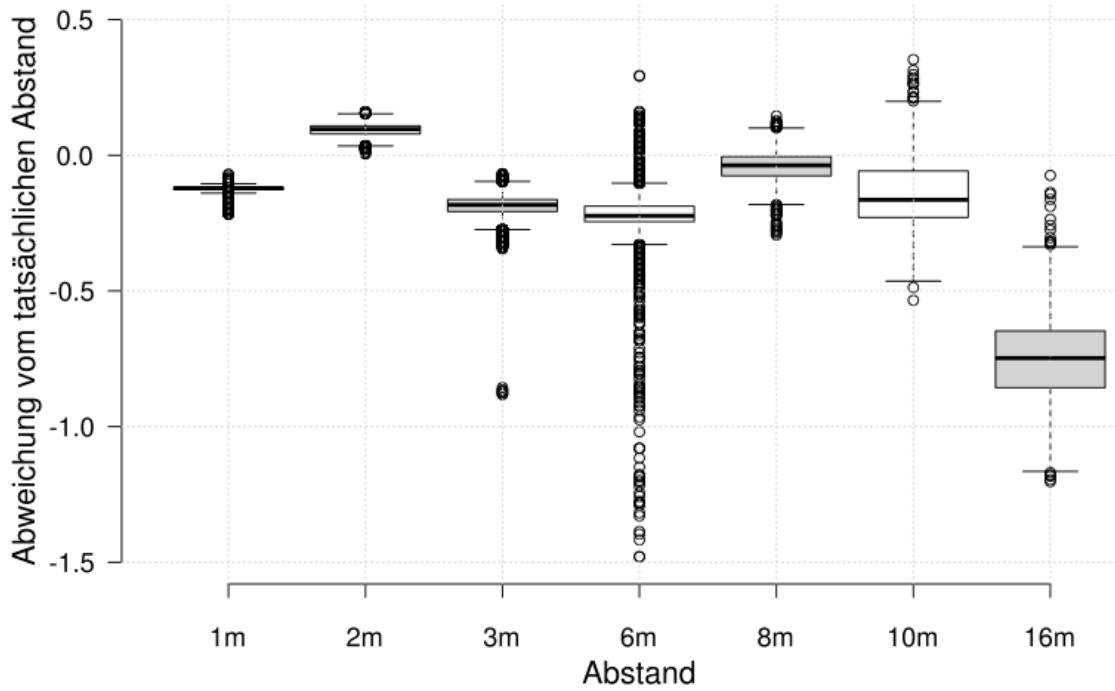


Abbildung 17: Boxplot-Diagramm der Punktestreuung für Scans aus verschiedenen Distanzen.

Es ist jeweils die Differenz von Depth-API-Abstand und realem Abstand gegeben. Ausreißer sind solche Werte, die weiter als das 1,5-fache vom Interquartilabstand über dem dritten oder unter dem ersten Quartil liegen.

**Abweichung der Distanz** Zusätzlich zu den Eigenschaften Planarität und Oberflächenvarianz wurde die Abweichung der per Depth API bestimmten Distanz von der tatsächlichen Distanz in dieser Versuchsreihe bestimmt. In Abbildung 17 ist ein Diagramm der Abweichung der Distanzen der Punkte zu sehen, in Abhängigkeit vom Scanabstand. Durch die Darstellung als Boxplot-Diagramm kann die Verteilung der Punkte gut erkannt werden. Es fällt auf, dass die Streuung ab einer Distanz von 3m merklich zunimmt. Der Bereich, in dem Punkte vorkommen können, wird breiter. Aber auch der Bereich, in dem sich die Hälfte aller Punkte befindet (breit gekennzeichnet) wird mit zunehmendem Abstand größer, was bedeutet, dass nicht nur eine breitere Streuung bei Ausreißern auftritt, sondern auch die Streuung aller Punkte zusammen zunimmt. Dies bestätigt die Ergebnisse der Analyse von Planarität und Oberflächenvarianz. Interessant ist außerdem, dass die Distanzen, mit Ausnahme des Scans aus 2m Entfernung, alle überwiegend kürzer als die realen Distanzen gemessen wurden. Der Wert für den Scan aus 6m Entfernung sticht mit besonders weit auseinander liegenden Ausreißern heraus. Eine besondere Ursache dafür ist nicht bestimmbar, der Kamerapfad des Scans weist keine Besonderheit auf und auch die Lichtverhältnisse waren für diesen Scan nicht verändert. Es wurde jedoch eine größere Pause (ca. 5 Minuten) zwischen dem Scan aus 3m und dem Scan aus 6m Entfernung gemacht, da auf eine freie Straße gewartet werden musste. Ein direkter Zusammenhang dazu ist zwar nicht auszuschließen, aber auch nicht

begründbar, weshalb die vielen Ausreißer als Anomalie klassifiziert werden können. Scans aus  $10m$  und  $16m$  Entfernung, wo eher ganze Hauswände und nicht nur die unteren Stockwerke erreicht werden können, weisen eine breite Streuung der Werte von fast  $1m$  auf. Für den Scan aus  $16m$  Entfernung fällt schließlich auf, dass die Entfernung als deutlich kürzer (durchschnittlich  $70cm$  und nicht einmal ein Ausreißer bei der richtigen Distanz) als der tatsächliche Abstand bestimmt wurden.

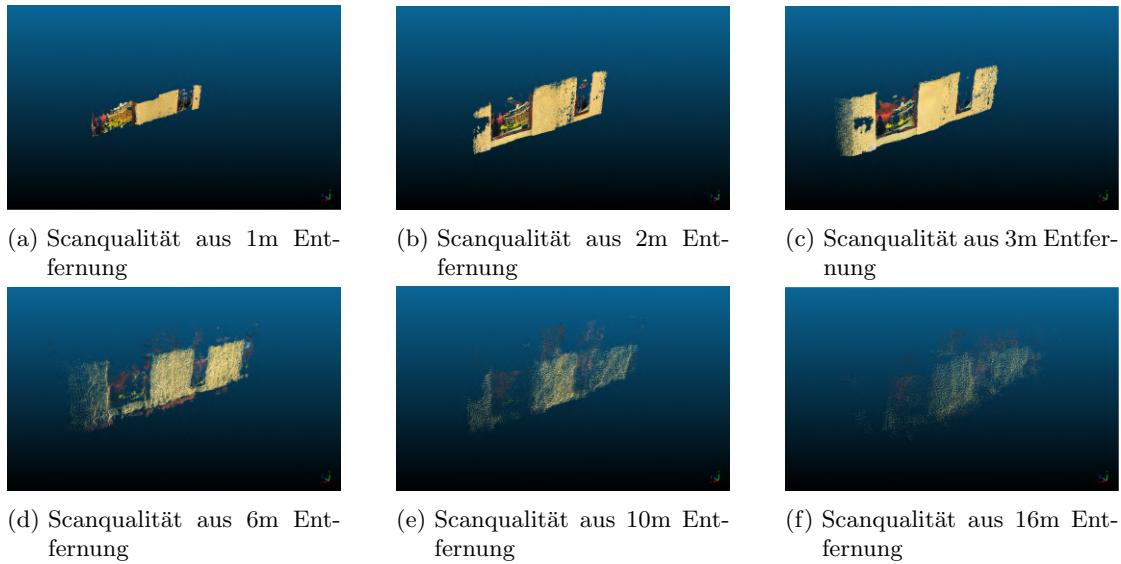


Abbildung 18: Scanqualität aus den Entfernungen  $1m$  bis  $16m$  bei der gleichen Wand.

**Visuelle Inspektion der Punktfolge** Auch bei manueller Betrachtung der in diesem Versuch untersuchten Wand können die oben quantifizierten Effekte gut beobachtet werden. In Abbildung 18 sind die Scans der Wand aus unterschiedlichen Entfernungen zu sehen. Das reduzierte Sichtfeld beim Scan aus  $1m$  Entfernung fällt sofort auf. Zwar wurden alle Scans auf einen ausgewählten Bereich zugeschnitten, aber der Scan aus nächster Nähe war noch kleiner (in der Höhe) als dieser Bereich. Außerdem kann in den Punktfolgen die mit Scandistanz zunehmende Streuung sehr gut beobachtet werden. Gleichzeitig ist hier aber auch zu sehen, dass die Dichte der Punktfolge mit zunehmender Scandistanz abnimmt.

### 5.3 Versuch 2: Horizontale Genauigkeit der Geospatial API



(a) StreetView-Ansicht der „Teststrecke Süd“



(b) Aktuelle Häuserfront der „Teststrecke Süd“.

Abbildung 19: Vergleich zwischen der StreetView-Ansicht und der aktuellen Häuserfront der „Teststrecke Süd“. Der Autohändler vom StreetView-Bild wurde durch ein Wohnhaus ersetzt.

**Zielsetzung:** Dieser Versuch soll klären, wie genau die horizontale Positionierung der Geospatial API ist, und ob das Alter von StreetView-Daten (insbesondere, wenn neue Häuser errichtet wurden) einen Einfluss auf die Genauigkeit hat. Gleichzeitig soll der reale Positionsfehler mit den durch die API angegebenen Genauigkeitswerten verglichen werden.

**Hypothese:** Gemäß Googles Angaben in der Dokumentation ist zu erwarten, dass eine 68-prozentige Wahrscheinlichkeit besteht, dass die gegebenen Genauigkeitswerte stimmen [10]<sup>9</sup>. Es ist auch zu erwarten, dass die Genauigkeit abnimmt, wenn die StreetView-Daten von der aktuellen Häuserfront abweichen, zum Beispiel, wenn seit Aufnahme der Bilder neue Häuser gebaut wurden.

#### 5.3.1 Durchführung

Für den Test wurden zwei gerade Teststrecken mit circa 100m Länge abgelaufen. Die Teststrecken verlaufen entlang dem Übergang zwischen zwei Pflasterungsarten auf dem Bürgersteig. So konnte während dem Scan eine gerade Linie gehalten werden. Die Kamera zeigte während dem Scanvorgang senkrecht auf die gegenüberliegende Häuserfront in 22m Entfernung. Eine der Teststrecken („Teststrecke Nord“) nutzt eine Häuserfront, die sich seit den letzten StreetView-Aufnahmen der Strecke im Jahr 2009 wenig verändert hat, während die andere Teststrecke („Teststrecke Süd“) auf eine Häuserfront blickt, in der seit den StreetView-Aufnahmen im Jahr 2008<sup>10</sup> das

<sup>9</sup>Diese Angabe musste aus einer archivierte Kopie der Dokumentation entnommen werden, da Google seitdem die Angabe nur noch auf die Rotation bezieht.

<sup>10</sup>aktuellste StreetView-Aufnahmen der Strecke

alte Gebäude abgerissen und durch ein neues ersetzt wurde (siehe Abbildung 19). So kann bewertet werden, inwiefern veraltete StreetView-Daten einen Einfluss auf die Tracking-Genauigkeit haben.

Die Teststrecken wurden jeweils viermal mit der gleichen Scankonfiguration pro Strecke abgelaufen. Beide nutzten eine Scankonfiguration mit einem Depth-Limit von  $40m$ , und ansonsten den Standardeinstellungen (siehe Kapitel 4.1.3). Nach dem Scavorgang wurden der UTM-Postprocessor, sowie der KML-Postprocessor auf die Scans angewandt.

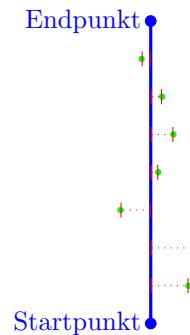


Abbildung 20: Auswertung der Positionsabweichung (rot) des Kamerapfads (grün) von der Ebene zwischen Start- und Endpunkt (blau). Beim Test wurde eine gerade Linie zwischen Start- und Endpunkt abgelaufen, weshalb die Ebene entlang des „Soll-Pfads“ verläuft und die roten Abstände die Positionsabweichung darstellen. (Aufsicht)

Die Kamerapfade in UTM Koordinaten (*frames\_utm.csv*-Datei) wurden mithilfe von *CloudCompare* geöffnet. Dort wurde eine senkrechte Ebene aus den Start- und Endkoordinaten der Scans konstruiert. Die Start- und Endkoordinaten wurden visuell auf *Google Maps* mit den Satellitenbildern bestimmt und in UTM umgewandelt. Bei der Erstellung der Ebene musste die Verschiebung, die *CloudCompare* beim Öffnen von Dateien auf die Koordinaten anwendet, beachtet werden (siehe Kapitel 4.1.9). Mit der Ebene konnte nun das *Cloud/Primitive distance*-Werkzeug verwendet werden, um die Distanz jedes Punktes zu der Ebene (also der „Soll-Position“) zu bestimmen (siehe auch Abbildung 20). Es gilt zu beachten, dass bei diesem Vorgehen nur die Abweichung entlang der Achse senkrecht zur Ebene untersucht werden kann.

Die Punktwolke mit den Distanzen als Skalarfeld wurde schließlich als CSV-Datei exportiert und in *Google Sheets* ausgewertet. Die Genauigkeitsangabe der Geospatial API, die Abweichung vom Soll-Pfad (also die Abstände zur Ebene), sowie die Differenz zwischen beiden Werten wurde in Abhängigkeit von der X-Koordinate (Northing) für jeden Scan in Diagrammen abgebildet.

### 5.3.2 Ergebnisse

Bei der Untersuchung der Kamerapfade über eine längere Strecke bei aktivem Tracking mittels Geospatial API konnten Erkenntnisse über die Genauigkeit, sowie die Genauigkeitsangabe der API gewonnen werden. Die Diagramme für je die Genauigkeitsangabe, die (absolute) Positions-



Abbildung 21: Diagramme zur horizontalen Genauigkeit der Geospatial API an zwei Teststrecken, eine ohne neue Bebauung seit der Aufnahme der StreetView-Bilder („Teststrecke Nord“, links) und eine mit neuer Bebauung („Teststrecke Süd“, rechts). Die x-Achse ist für Teststrecke Süd invertiert, da die x-Koordinate im Laufe des Tests kleiner wurde.

abweichung von der Soll-Position (also der Abstand zum abgegangenen Pfad), sowie die Differenz zwischen beiden Werten, sind in Abbildung 21 zu sehen. Die x-Achse für die „Teststrecke Süd“ ist dabei invertiert, da die x-Koordinate (entspricht UTM Northing) in diesem Test kleiner wurde, während sie bei der anderen Teststrecke größer wurde. Dies ist auf die unterschiedlichen Richtungen der Tests zurückzuführen („Teststrecke Nord“ Richtung Norden, „Teststrecke Süd“ Richtung Süden). Tabelle 3 listet die statistischen Werte *Durchschnitt* und *Standardabweichung* für jeweils die Genauigkeitsangabe und die absolute Positionsabweichung beider Teststrecken, sowie die maximale Positionsabweichung.

Wert	Teststrecke Nord	Teststrecke Süd
Durchschnitt Genauigkeitsangabe	0,64m	0,64m
Standardabweichung Genauigkeitsangabe	0,11m	0,09m
Durchschnitt Positionsabweichung	0,12m	0,32m
Standardabweichung Positionsabweichung	0,10m	0,18m
Maximale Positionsabweichung	0,51m	1,03m

Tabelle 3: Statistische Kennwerte der Genauigkeitsangabe und Positionsabweichung der Tests für horizontale Genauigkeit.

Zunächst fällt auf, dass sich die Genauigkeitsangaben, trotz unterschiedlich aktueller StreetView-Daten, im selben Bereich bewegen. Mit einem Durchschnitt von  $64\text{cm}$  für die Genauigkeitsangabe bei beiden Teststrecken, sowie einer Standardabweichung der Genauigkeitsangabe im ungefähr selben Bereich ( $\pm 2\text{cm}$ ) gibt es keinen merkbaren Unterschied beim Wert der Genauigkeitsangaben. Auch bei visueller Inspektion der Diagramme der Genauigkeitsangabe bei beiden Teststrecken (siehe Abbildung 21a und 21b) fällt auf, dass sich die Werte bis auf einen Ausreißer bei der „Teststrecke Nord“ im gleichen Bereich bewegen. Es ist jedoch auch anzumerken, dass sich die Genauigkeitsangabe bei der „Teststrecke Süd“ an sehr regulären Punkten in einer für alle vier Scandurcläufe ähnlichen Art ändert. Dieselbe Regelmäßigkeit kann für die „Teststrecke Nord“ nicht beobachtet werden. Dies legt die Schlussfolgerung nahe, dass der Effekt damit zusammenhängt, dass die StreetView-Daten bei „Teststrecke Süd“ nicht mehr der tatsächlichen Bebauung entsprechen. Da die Werte für alle Scans an den gleichen Orten sinken, kann zunächst vermutet werden, dass sich an den Orten visuelle Features befinden, an denen VPS eine Position bestimmen kann und damit genauer wird. Ein direkter Zusammenhang zu Objekten an den Positionen kann jedoch nicht bestimmt werden, weshalb diese These schwer zu beweisen ist. Die Regelmäßigkeit der Abstände zwischen den Tälern spricht auch gegen diese These und eher für ein Positionsupdate aus anderen Quellen (z.B. GPS), welches alle  $10\text{m}$  (ungefährer Abstand zwischen den Tälern) von der Geospatial API ausgelöst wird, um fehlende visuelle Positionierung durch den VPS auszugleichen.

Bei Betrachtung der Positionsabweichung (siehe Abbildung 21c und 21d) kann ein deutlicher Unterschied zwischen beiden Teststrecken identifiziert werden. Während die Position bei der „Teststrecke Nord“ im Schnitt um nur  $12\text{cm}$ , mit einem Maximum von  $51\text{cm}$ , abweicht, ist die durchschnittliche Positionsabweichung für die „Teststrecke Süd“ mit  $32\text{cm}$  und einem Maximum

von  $1,03m$  deutlich höher. Dies entspricht der Erwartung für diesen Test und weist nach, dass nicht aktuelle StreetView-Daten einen direkten Einfluss auf die Genauigkeit der Positionierung mit der Geospatial API haben. Dennoch bleibt der Fehler in einem relativ geringen Rahmen, verglichen mit bisherigen Methoden zur globalen Smartphone-Ortung.

Zuletzt wurde die Differenz zwischen den Genauigkeitsangaben und der tatsächlichen Positionsabweichung gebildet (siehe Abbildung 21e und 21f). Ein negativer Wert bedeutet hier, dass die tatsächliche Positionsabweichung größer als die Genauigkeitsangabe war. Dies ist nur bei der „Teststrecke Süd“, und auch nur an zwei Stellen der Fall. In den meisten Situationen ist die Positionsabweichung geringer als die angegebene Genauigkeit.

## 5.4 Versuch 3: Übereinstimmung von Scans am gleichen Ort

**Zielsetzung:** In diesem Versuch soll bestimmt werden, wie die Genauigkeit der Depth API und der Geospatial API zusammenspielen, wenn ein gleiches Stück einer Hauswand gescannt wird. Dies ist besonders relevant, wenn Punktfolgen aus mehreren Scans zusammengesetzt werden, was bei Scans von Straßenzügen aufgrund der potenziellen Größe dieser Scans durchaus notwendig sein kann. Aufgrund der starken Unterschiede bei der Genauigkeit und Streuung der Tiefenerkennung aus größerer Distanz (siehe Versuch 1) soll dieser Test einmal für Scans aus der Nähe und einmal aus Entfernung durchgeführt werden.

**Hypothese:** Es ist zu erwarten, dass die Abweichung zwischen den Punktfolgen der in Versuch 2 bestimmten Ungenauigkeit der Position der Geospatial API entspricht. Außerdem wird die in Versuch 1 bestimmte Streuung der Punkte bei Scans aus größerer Entfernung für zusätzliche Abweichungen sorgen, sodass die Positionsfehler der Geospatial API weniger relevant sind und in der Streuung untergehen.

### 5.4.1 Durchführung

Für die Untersuchung in diesem Versuch soll eine Hauswand aus einer nahen Entfernung ( $3m$ ) und von der anderen Straßenseite ( $16m$ ) gescannt werden. Dabei soll der Scan fünfmal pro Position wiederholt werden. So kann im Anschluss die „Passform“ der Scans, also wie sehr sich die unterschiedlichen Abläufe in ihrer Position unterscheiden, bewertet werden.

Die Scanergebnisse werden visuell in *CloudCompare* ausgewertet, sowie mit dem Werkzeug *Cloud/Cloud distance*. Dieses erlaubt es anhand von optischen Features die Distanz zwischen zwei Punktfolgen zu bestimmen. Dabei werden der durchschnittliche Abstand, sowie die Standardabweichung angegeben. Das Tool soll für jede mögliche Kombination der fünf Scans pro Scanabstand ausgeführt werden, um die durchschnittlichen Abweichungen zwischen allen Scans zu bestimmen. Dies ist erforderlich, da keine Grundwahrheit für die Punktfolgen bekannt ist. Es kann lediglich die Variation zwischen den Scan betrachtet werden.

### 5.4.2 Ergebnisse

Scan Entfernung	Verglichene Scans	Durchschnittlicher Abstand	Standardabweichung Abstand
3m	Scan 0 - Scan 1	0.08m	0.06m
	Scan 0 - Scan 2	0.03m	0.03m
	Scan 0 - Scan 3	0.11m	0.05m
	Scan 0 - Scan 4	0.03m	0.04m
	Scan 1 - Scan 2	0.1m	0.08m
	Scan 1 - Scan 3	0.04m	0.04m
	Scan 1 - Scan 4	0.07m	0.06m
	Scan 2 - Scan 3	0.11m	0.06m
	Scan 2 - Scan 4	0.03m	0.04m
	Scan 3 - Scan 4	0.09m	0.06m
16m	Min	0.03m	0.03m
	Max	0.11m	0.08m
	Durchschnitt	0.069m	0.052m
16m	Scan 0 - Scan 1	0.19m	0.16m
	Scan 0 - Scan 2	0.23m	0.22m
	Scan 0 - Scan 3	0.14m	0.17m
	Scan 0 - Scan 4	0.53m	0.37m
	Scan 1 - Scan 2	0.14m	0.18m
	Scan 1 - Scan 3	0.09m	0.14m
	Scan 1 - Scan 4	0.56m	0.25m
	Scan 2 - Scan 3	0.09m	0.1m
	Scan 2 - Scan 4	0.42m	0.19m
	Scan 3 - Scan 4	0.42m	0.28m
16m	Min	0.09m	0.1m
	Max	0.56m	0.37m
	Durchschnitt	0.281m	0.206m

Tabelle 4: Abstände zwischen Punktwolken am gleichen Ort

In Tabelle 4 können die durchschnittlichen Distanzen zwischen verschiedenen Punktwolken derselben Hauswand betrachtet werden. Dies beschreibt, wie gut Scans desselben Objektes zueinander passen, beziehungsweise wie weit das gleiche Feature in verschiedenen Scan-Durchläufen verschoben sein kann. Für Scans aus *3m* Entfernung beträgt die maximale Distanz zwischen den Wolken *11cm*. Dies entspricht ungefähr der in Versuch 2 bestimmten, durchschnittlichen horizontalen Ungenauigkeit der Geospatial API, und damit wurde dieser Teil der Hypothese bestätigt.

Bei Scans aus *16m* Entfernung hingegen beträgt die maximale Distanz zwischen Scans *56cm*. Diese Abweichung ist nicht mehr allein durch die Ungenauigkeit der Geospatial API erklärbar und eher auf leichte Variationen in der Tiefenerkennung mit der Depth API zurückzuführen. Die höhere Streuung der Scans aus dieser Entfernung (vgl. Versuch 1 in Kapitel 5.2.2) hat vermutlich auch einen Einfluss auf diesen Wert. Damit bestätigt auch dieses Ergebnis die Hypothese.

Bei visueller Analyse der Scans fällt insbesondere die Streuung der Punkte aus größerer Distanz auf. Oberflächen wie Häuserfronten sind als solche erkennbar, aber sind durch die Streuung der Tiefenerkennung auf größere Distanzen sehr breit gestreut, die Wände wirken dementsprechend „dick“ (siehe Abbildung 22b). In Abbildung 22c ist der Positionsunterschied der Hauswand von beiden Scans sehr gut zu erkennen. Die Streuung des weit entfernten Scans verläuft eher nach rechts (die Richtung, aus der gescannt wurde), verglichen mit der Wand vom nahen Scan, was das Ergebnis aus Versuch 1 bestätigt, wonach die Distanzen eher zu kurz gemessen werden.

Auch die Störung der Scans durch Bäume ist in dem Scan aus *16m* Entfernung gut sichtbar. In Abbildung 22b kann man sehen, wie Teile der Hauswand (grau) durch den Baum als weiter vorne liegend erkannt werden. Die Position des Baumes ist auch in Abbildung 22a sichtbar.



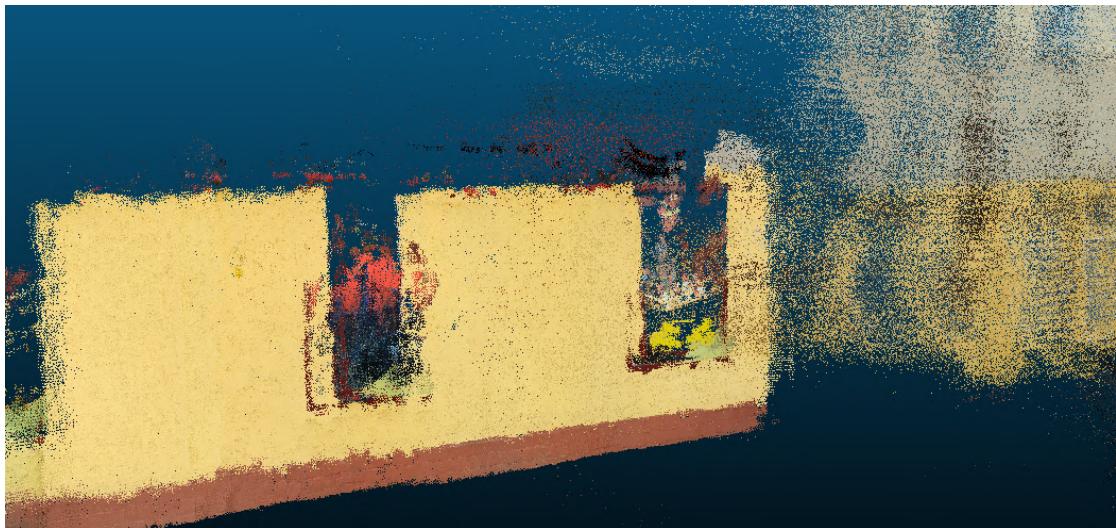
(a) Frontansicht des Scans aus 16m Entfernung



(b) Seitenansicht des Scans aus 16m Entfernung. Die Wand rechts ist ungefähr 1m dick gestreut



(c) Vergleich des Scans aus 3m (vorne bzw. unten im Bild) und 16m (hinten bzw. oben im Bild) Entfernung, Seitenansicht



(d) Vergleich des Scans aus 3m (links im Bild) und 16m (rechts im Bild) Entfernung, Frontansicht

Abbildung 22: Ansichten der Scans aus 3m und 16m Entfernung. Es ist der Einfluss des Baumes (Abbildung 22b, links im Bild, steht aus der Wand hervor), sowie der Vergleich der Wanddicke zum Scan aus 3m Entfernung (Abbildungen 22c und 22d) sichtbar.

Die Fehler in der Positionierung der Scans sind auch gut sichtbar. So kann in Abbildung 22d ein „doppelter“ Fensterrahmen erkannt werden. Dieser kommt durch die leichten Positionierungsfehler zustande, welche zu Beginn dieses Kapitels quantifiziert wurden.

Die Fehler durch die Geospatial API fallen eher bei Scans aus nächster Nähe auf, bei Scans aus größerer Entfernung gehen sie in der Streuung der Depth API unter.

## 5.5 Versuch 4: Positionsabweichung bei weit entferntem Anker

**Zielsetzung:** Mit dieser Versuchsreihe soll die Positionsgenauigkeit ohne kontinuierliche Georeferenz (also nur mit einer Georeferenz des Start-Ankers) auf weite Distanz getestet werden. Diese Positionsgenauigkeit wird durch die Rotationsgenauigkeit der in initialen Georeferenz, sowie die Genauigkeit des relativen ARCore Trackings bestimmt.

**Hypothese:** Mit einer kurzen Rechnung kann bestimmt werden, dass bei einem initialen Rotationsfehler von  $2^\circ$  die Position in einer Entfernung von  $200m$  vom Ankerpunkt um ca.  $\tan(2^\circ) \cdot 200m \approx 7m$  abweicht. Die Genauigkeitsangabe der Geospatial API war bei bisherigen Tests immer  $< 2^\circ$ , sodass, vorausgesetzt die Genauigkeitsangabe ist korrekt, von einem Fehler von ca.  $7m$  auf  $200m$  Strecke ausgegangen werden kann. In [60] wurde die Abweichung von ARCores relativem Positionstracking auf durchschnittlich  $2cm$  pro Meter bestimmt. Hochgerechnet auf  $200m$  entspricht dies einer Abweichung von  $4m$ . Somit kann durch den Rotationsfehler des Ankers und diesem Fehler im relativen Tracking von einer Abweichung von bis zu  $11m$  bei einer Entfernung von  $200m$  vom Ankerpunkt ausgegangen werden.

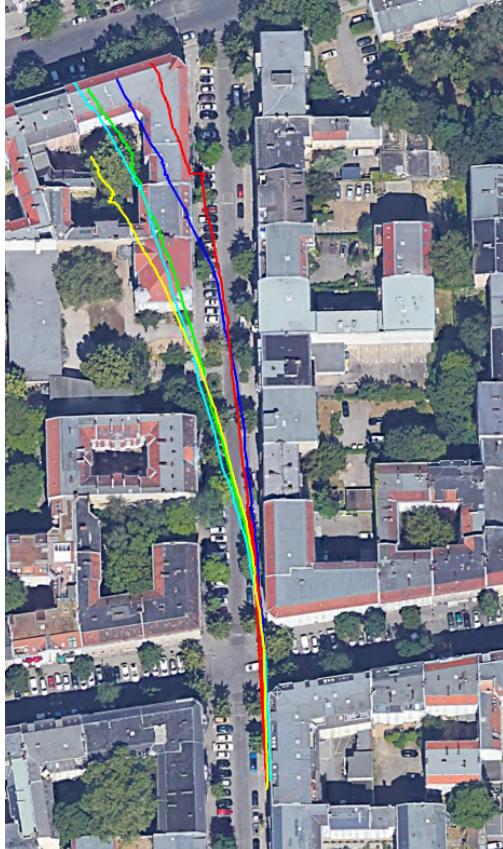
### 5.5.1 Durchführung

Für den Test soll mit einer Scankonfiguration ohne kontinuierliche Georeferenz, einem Tiefenlimit von  $40m$  und sonst den Standardparametern (siehe Kapitel 4.1.3) auf einer Strecke von  $200m$  mehrmals gescannt werden. Dabei soll ein Bürgersteig abgelaufen werden, mit Blick auf die gegenüberliegende Straßenseite.

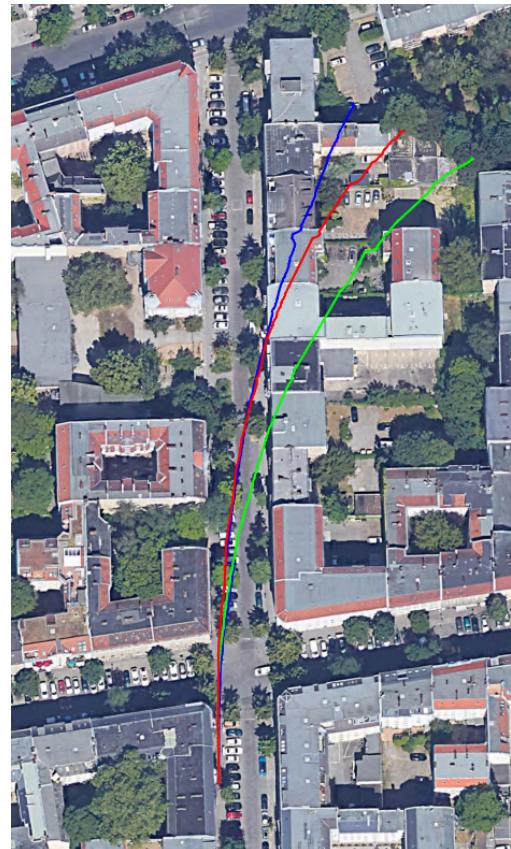
Die Kamerapfade sollen nur mit *Google My Maps* analysiert werden. Das dort vorhandene Werkzeug zur Distanzmessung soll genutzt werden, um die Abweichung der Position in  $200m$  Entfernung vom initialen Anker zu bewerten.

### 5.5.2 Ergebnisse

Die Kamerapfade wichen entgegen den Erwartungen deutlich weiter als  $11m$  von der Zielposition ab ( $25 - 60m$ ). Lediglich für die ersten  $50m$  waren die Positionen noch innerhalb von  $1m$  Abweichung. Danach steigt diese immer weiter an und der Kamerapfad beschreibt statt einer geraden Linie eher eine Kurve. Dies spricht dafür, dass der größte Teil der Abweichung nicht aus dem Rotationsfehler des initialen Ankers stammt, sondern aus einem Fehler im relativen Positionstracking von ARCore.



(a) Kamerapfade fünf unterschiedlicher Scans entlang der gleichen 200m Strecke bei Tracking relativ zu einem initialen Anker. Es wurde der östliche Gehweg genutzt und das Smartphone war Richtung Westen ausgerichtet.



(b) Kamerapfade drei unterschiedlicher Scans entlang der gleichen 200m Strecke bei Tracking relativ zu einem initialen Anker. Dabei wurde der Testablauf aus Abbildung 23a gespiegelt, d.h. es wurde der westliche Gehweg genutzt und das Smartphone war Richtung Osten ausgerichtet.

Abbildung 23: Kamerapfade ohne kontinuierliche Georeferenz (mit Tracking relativ zu einem initialen Anker)

Interessant ist, dass die Kamerapfade alle in dieselbe Richtung abweichen, nämlich in die Blickrichtung (siehe Abbildung 23a). Deshalb wurden drei weitere Scans auf der gegenüberliegenden Straßenseite mit entgegengesetzter Blickrichtung durchgeführt, um zu prüfen, ob diese Abweichung tatsächlich von der Blickrichtung abhängig ist. Tatsächlich ist die Kurve für diese drei Scans dann auch invertiert (siehe Abbildung 23b). Dies legt nahe, dass es beim ARCore Positionstracking relativ zu einem Anker einen Blickrichtung-abhängigen Fehler gibt. Weitere Untersuchungen dazu sind nicht Teil dieser Arbeit, aber bieten einen guten Ansatzpunkt für weitergehende Forschung zur Bestimmung und Mitigation des Fehlers (vgl. Kapitel 8).

Ergebnis dieses Versuchs ist daher, dass Scans ohne kontinuierliche Georeferenz nur auf kurze Distanz ( $< 50m$ ) eine halbwegs genaue Position besitzen und auf längere Distanzen stark abweichen ( $> 50m$  Abweichung auf  $200m$  Strecke).

## 5.6 Sonstige Beobachtungen während der Entwicklung

Während der Entwicklung der App konnten weitere Beobachtungen gemacht werden, für die keine dedizierten Tests durchgeführt wurden. Es handelt sich daher auch nur um tendenzielle, nicht quantifizierte Aussagen.

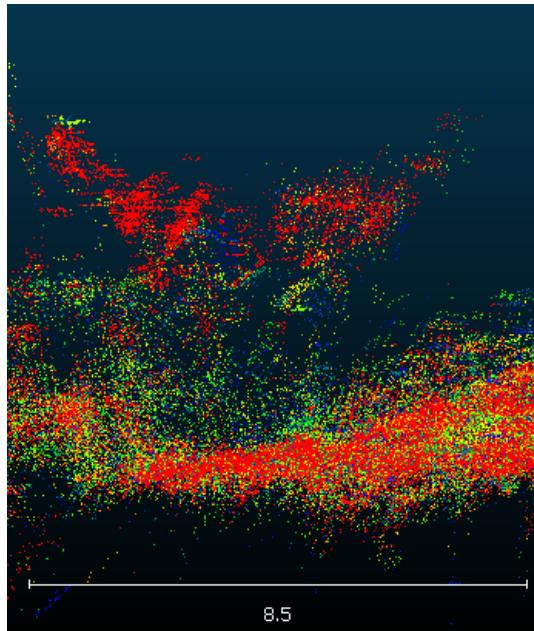


Abbildung 24: Aufsicht auf die Streuung der Confidence-Werte (rot = hohe Confidence, blau = niedrige Confidence). Vor einer Wand (unten im Bild) liegen Ausreißer mit sehr hoher Confidence (oben im Bild) und in der Wand sind die Confidence-Werte stark gestreut. Der Maßstab ist in Metern angegeben.

**Confidence-Werte** Die erste Beobachtung bezieht sich auf die (normalisierten) Confidence-Werte der Depth-API. Erwartungsgemäß sollte dieser Wert für richtige Punkte höher sein als für gestreute Punkte. In der Praxis hingegen wurde schnell klar, dass diese Werte sehr breit verteilt sind, und viele falsche Punkte eine hohe Confidence erhalten, während viele richtige Punkte nur eine niedrige Confidence besitzen (siehe Abbildung 24). Ein Muster war für diese Punkte nicht erkennbar, weshalb dieser Wert in der Praxis nicht genutzt werden konnte, um die Punktwolken durch einen einfachen Filter zu verbessern. Es konnte lediglich eine grobe Tendenz ermittelt werden, wonach Ausreißer ( $> 5m$  Fehler) in den Punktwolken eher eine Confidence unter 0,5 besitzen. Deshalb wurde 0,5 als Standardwert für den Confidence-Cutoff (vgl. Kapitel 4.1.3) festgelegt und auch für die Versuche genutzt. Es ist jedoch zu beachten, dass so nicht nur die Ausreißer, sondern auch viele richtige Punkte ausgeschlossen wurden.

**Bewegung vor Scanbeginn** Eine weitere besondere Auffälligkeit war, dass die Tiefenerkennung für die ersten Frames eines Scans besonders schlecht war, wenn vor Scanbeginn keine Bewegung des Smartphones zur Seite oder auf das

Objekt zu bzw. vom Objekt weg, erfolgte. Dies ist auf die Funktionsfähigkeit der Depth API zurückzuführen, da verschiedene Perspektiven, also eine Bewegung des Geräts, erforderlich sind, um ein korrektes Tiefenbild zu erhalten. Eine solche Bewegung vor Scanbeginn verhindert falsch erkannte Punkte in der resultierenden Punktwolke. Deshalb wurde bei allen Versuchsdurchläufen, wo die Punktwolke relevant war, eine solche initiale Bewegung von ca. 1m seitwärts vor Scanbeginn durchgeführt.

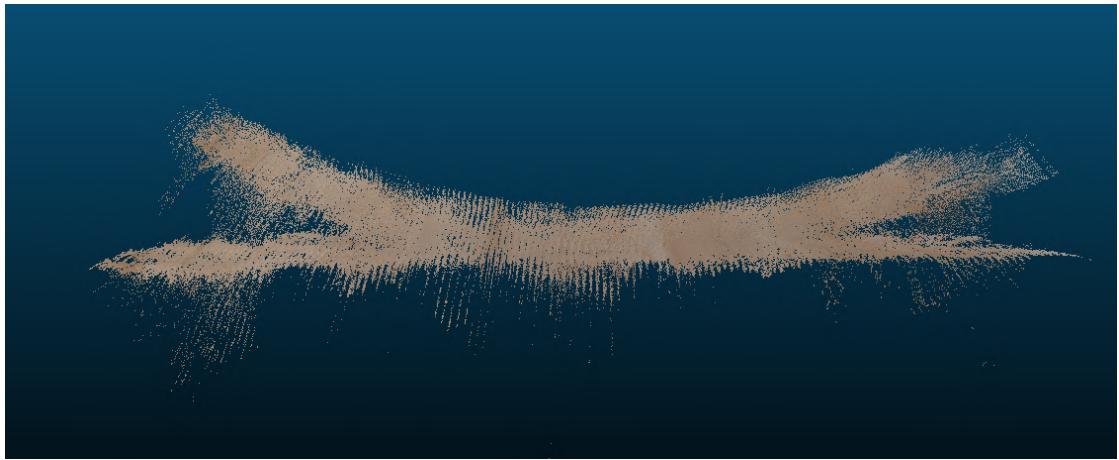


Abbildung 25: Test für Referenz der in der Depth API angegebenen Tiefe.

**Referenz für Tiefenwert** Da in der Dokumentation der Depth API nicht spezifiziert wurde, ob der angegebene Tiefenwert eines Pixels die Distanz zur Linsenebene ( $d_1$  in Abbildung 26), oder die Distanz zur Linse der Kamera ( $d_2$  in Abbildung 26) ist, wurde ein kleiner Test zur Überprüfung dieser Referenz durchgeführt. Diese Referenz ist nämlich zur Berechnung der Position eines Pixels in der Welt relevant (siehe auch Kapitel 2.4). ToF-Sensoren und andere Tiefensensorik gibt diese Tiefe normalerweise zum Sensor oder zur Kamera an, nicht zur Linsenebene. Zur Überprüfung, welche Tiefe die Depth API nutzt, wurde mit dem Smartphone senkrecht auf einen ebenen Boden aus 1m Abstand ein Scan durchgeführt, einmal unter Annahme der Tiefe zur Kameralinse, und einmal unter Annahme der Tiefe zur Linsenebene. Der Scan mit der Tiefe zur Linsenebene lieferte einen ebenen Boden, während der Scan unter Annahme der Tiefe zur Kamera einen gekrümmten Boden geliefert hat (siehe Abbildung 25). Somit wurde bestimmt, dass die Tiefenangabe der Depth API die Distanz des Objekts zur Linsenebene und nicht zur Kameralinse ist, entgegen der für Tiefensensoren üblichen Tiefe zum Sensor bzw. zur Linse.

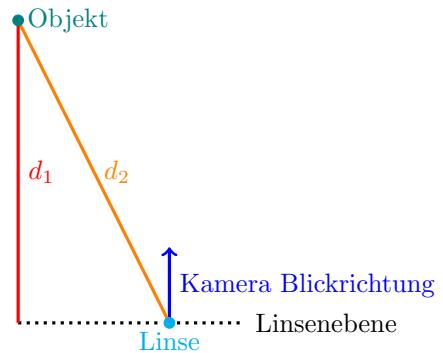


Abbildung 26: Unterschiedliche Interpretationen der Tiefe eines Objekts, entweder zur Linse oder zur Linsenebene (Aufsicht)

**Framerate** Während der initialen Entwicklung der App wurde die Berechnung globaler Koordinaten noch während dem Scavorgang durchgeführt. Diese Operation und die Umrechnung zwischen Tiefenbild- und CPU-Bild-Koordinaten reduzierten die Framerate der Scans signifikant. Nach Behebung dieser Performanceprobleme durch Verschiebung der Berechnung globaler Koor-

dinaten in einen Nachbearbeitungsprozess und Optimierung der Bild-Koordinaten-Umrechnung mit normalisierten Koordinaten, konnte beobachtet werden, dass nicht nur die Framerate beim Scannen verbessert wurde, sondern dass die Qualität der Tiefenbilder zunahm. Es kann also vermutet werden, dass die Berechnung von Tiefe durch die Depth API unter kostspieligen Operationen für Kameraframes leidet. Die Entwicklung der Hilfsklasse `TimingHelper` erlaubte die Zeitmessung der unterschiedlichen Prozesse und Identifikation der oben erwähnten teuersten Operationen.

**Gehgeschwindigkeit** Die Tiefenerkennung war in den Versuchen sehr abhängig von der Gehgeschwindigkeit und man musste darauf achten, sich unnatürlich langsam<sup>11</sup> zu bewegen, da ansonsten keine neuen Tiefenbilder erzeugt wurden. Dieser Effekt konnte unabhängig von Scanabstand beobachtet werden.

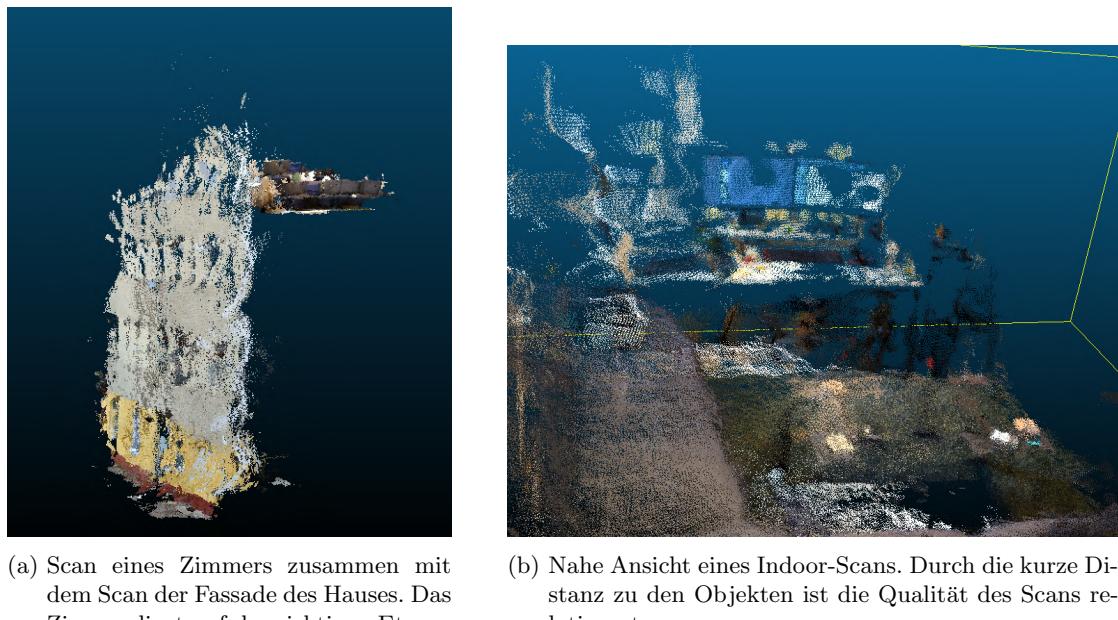


Abbildung 27: Kombination aus Indoor- und Outdoor-Scans

**Kombination von Indoor- und Outdoor-Scans** Die Fähigkeit der App, Scans ohne kontinuierliche Georeferenz (mit nur einer Georeferenz am Start-Anker) aufzunehmen (vgl. Kapitel 4.1.3), erlaubt es, Indoor-Scans mit Georeferenz aufzunehmen, indem die initiale Georeferenz durch ein Fenster erhalten wird. Diese georeferenzierten Scans können dann zusammen mit Outdoor-Scans betrachtet oder sogar fusioniert werden, um sowohl Fassaden als auch Innenräume zusammen betrachten zu können, oder sehr detaillierte Modelle von Gebäuden zu erhalten (siehe Abbildung 27).

<sup>11</sup>deutlich langsamer als „normale“ Gehgeschwindigkeit

**Höhenerkennung der Geospatial API** Die Höhenerkennung der Geospatial API wurde in keinem dedizierten Versuch untersucht. Es konnte bei Tests zu Indoor-Scans jedoch festgestellt werden, dass die Position des Smartphones manchmal auf Straßenhöhe eingeordnet wird, statt auf der tatsächlichen Höhe im Gebäude (ca. 10m Höhenunterschied, siehe Abbildung 28). Dies deutet darauf hin, dass die Geospatial API einen Bias dazu hat, das Smartphone auf Straßenhöhe zu lokalisieren, da Höhenfehler in diesem Ausmaß bei Scans auf der Straße nicht auftraten.

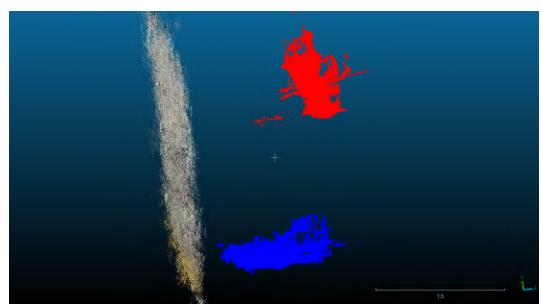


Abbildung 28: Höhe eines Indoor Scans (blau) ist nicht korrekt (sollte eigentlich auf einer Höhe mit dem roten Scan sein)

## 6 Diskussion

### 6.1 Zusammenfassung der Ergebnisse

Mit den Untersuchungen dieser Arbeit wurde die erwartbare Genauigkeit der mit der App erstellten Punktwolken bestimmt. Es wurde der Einfluss der Depth API, die horizontale Genauigkeit der Geospatial API, die Übereinstimmung von Scans am gleichen Ort, sowie die Positionsabweichung ohne kontinuierliche Georeferenz bestimmt.

In Versuch 1 wurde die Scanqualität in Abhängigkeit vom Abstand zu einem Objekt bestimmt. Bei Scans aus kurzer Distanz ( $< 3m$ ) wurden dabei sehr gute Ergebnisse erzielt, mit geringer Streuung ( $\pm 5cm$ ) und geringem Tiefenfehler (ca.  $10cm$ ) für die Hälfte der Punkte. Auch Ausreißer liegen hier noch in einem kleinen Rahmen. Features der Wand mit einem Tiefenunterschied von mindestens  $10cm$  sind bei dieser Entfernung als solche in der resultierenden Punktwolke erkennbar und gehen nicht in der Streuung unter. Bei größerer Entfernung ( $6m - 16m$ ) nimmt die Streuung, sowie der Tiefenfehler stark zu. Bei Scans aus  $16m$  Entfernung haben die Hälfte aller Punkte eine Streuung ca.  $25cm$  und einen Tiefenfehler von  $70cm - 90cm$ . Features brauchen bei dieser Distanz einen Tiefenunterschied von  $80cm$ , um als solche noch erkennbar zu sein.

In Versuch 2 wurde die horizontale Genauigkeit der Geospatial API über längere Distanz mit aktuellen und nicht aktuellen StreetView-Daten untersucht. Die durchschnittliche Positionsabweichung lag dabei bei aktuellen Daten im Rahmen von  $\pm 12cm$  und bei nicht aktuellen Street-View Bildern bei  $\pm 32cm$ . In den meisten Fällen blieb die Positionsabweichung dabei im Rahmen der durch die API angegebenen Genauigkeit, weshalb diese zum Identifizieren und Ausschließen ungenauer Positionen geeignet ist.

In Versuch 3 wurde die Wiederholbarkeit der Scans an gleichen Orten für zwei unterschiedliche Scanentfernungen untersucht. Hierbei wurde bei Scans aus  $3m$  Entfernung ein durchschnittlicher Unterschied von ca.  $7cm$  zwischen den Scans bestimmt, mit einem Maximum von  $11cm$ . Bei Scans aus  $16m$  Entfernung hingegen betrug die Abweichung zwischen Scans im Schnitt  $28cm$ , mit einem Maximum von  $56cm$ . Es erfolgte außerdem eine visuelle Analyse, bei der die Scans beider Distanzen miteinander verglichen wurden. Hierbei fiel insbesondere der Unterschied in der Streuung der Punkte auf, die bei Scans aus weiter Entfernung deutlich größer war. Es wurde außerdem identifiziert, dass Bäume vor einer Hauswand einen starken Einfluss auf die resultierende Punktwolke haben und die Hauswand „mitnehmen“, sie also an die Position der Baumkrone verschieben.

In Versuch 4 sollte der Rotationsfehler der Geospatial API und der Fehler des relativen Positionstrackings von ARCore bestimmt werden. Die Abweichung von ca.  $50m \pm 20m$  auf eine gradlinige Strecke von  $200m$  liegt dabei deutlich über der Erwartung von  $11m$ . Da eine Kurve statt einer geraden Linie entstanden ist, stammt der Großteil des Fehlers aus dem Positionstracking von ARCore relativ zu einem Anker und nicht aus dem Rotationsfehler des initialen Ankers. Genauer konnte bestimmt werden, dass dieser Fehler von der Blickrichtung abhängig ist. Die Kurve ist dabei in die Blickrichtung der Kamera gekrümmmt.

## 6.2 Interpretation der Ergebnisse

Unter Betrachtung der Tatsache, dass allein durch die Höhe von Hauswänden die Distanz zum Smartphone schnell  $10m$  oder mehr betragen kann, bedeuten die Ergebnisse aus Versuch 1, dass Hauswände insgesamt nur grob mit der App gescannt werden können, mit relativ starker Streuung bei der Tiefenerkennung. Die Tiefe von Fenstern oder die Struktur von Verzierungen der Hauswand werden in der Struktur der Punktwolke nicht sichtbar, lediglich Strukturen mit großem Tiefenunterschied, wie zum Beispiel Balkone oder Einfahrten, können dreidimensional in der Punktwolke sichtbar sein. Durch die Farbwerte hingegen können die Positionen von Fenstern und größeren Verzierungen grob bestimmt werden. Wenn kurze Abstände zur Hauswand gehalten werden können, zum Beispiel, wenn nur das Erdgeschoss gescannt werden soll, so können auch sehr detaillierte Scans erstellt werden, in denen kleine Features dreidimensional aufgenommen werden und nicht in der Streuung der Tiefenerkennung untergehen. Dabei gilt es dann allerdings zu beachten, dass die Geospatial API in diesen Fällen weniger Daten im Bild zur Positionsbestimmung zur Verfügung hat.

Die Geospatial API, und damit der VPS, hat sich in den Tests, insbesondere Versuch 2, als sehr genau erwiesen. Der horizontale Fehler betrug im Schnitt lediglich  $12cm$  bei aktuellen StreetView-Daten und wurde selbst bei komplett veralteten StreetView-Daten nicht deutlich ungenauer und blieb immer im Rahmen von  $\pm 1m$ . Im Vergleich mit bisherigen globalen Ortungsmethoden, insbesondere GPS in urbanen Umgebungen, wo bisher nur Genauigkeiten im Bereich von  $5m - 10m$  ohne externe Hardware möglich waren [61, 62], ist die Positionierung mit der Geospatial API deutlich genauer. Ein weiterer Bonus sind die Genauigkeitsangaben, welche nach den Ergebnissen aus Versuch 2 meistens eine höhere Ungenauigkeit angeben als tatsächlich vorliegt. Es gilt jedoch zu beachten, dass veraltete StreetView-Daten die Genauigkeit negativ beeinflussen, dies aber nicht im Wert der Genauigkeitsangaben ablesbar ist.

Das Zusammenfügen von Scans funktioniert laut den Ergebnissen aus Versuch 3 relativ gut. Die Positionsfehler befinden sich meistens in einem Rahmen, in dem die Punktwolken laut Versuch 1 und 2 sowieso gestreut sind. Bei visueller Betrachtung fallen Artefakte der Positionsfehler kaum auf und innerhalb eines Scans sind die Fehler konsistent. Somit können mit der App dichtere Punktwolken erstellt werden, indem Bereiche mehrmals gescannt werden. Für Konsistenz in den Farbwerten ist jedoch darauf zu achten, dass die Lichtverhältnisse bei allen Scans vergleichbar sind. Es kann außerdem davon ausgegangen werden, dass unterschiedliche Scanabschnitte, zum Beispiel wenn unterschiedliche Straßen separat gescannt werden, mit einem relativ geringen Positionsfehler zusammenpassen. Bei der visuellen Inspektion der Wolken hat sich auch die Beobachtung aus Versuch 1 bestätigt, dass Punkte bei größerer Scandistanz eher näher als die tatsächliche Distanz erkannt werden.

Versuch 4 hat einen Blickrichtung-abhängigen Fehler im relativen Positionstracking von AR-Core aufgezeigt. Dieser bedeutet für die Anwendung der App, dass Scans ohne kontinuierliche Georeferenz nur auf eine Distanz von ca.  $50m$  relativ genau bleiben und ab da von diesem starken Drift zunehmend betroffen sind. Es ist zu erwarten, dass der Drift deutlich geringer ist, während

der Anker nah am Smartphone und im Bild ist, da Anker von ARCore kontinuierlich angepasst werden, sodass sie an derselben Stelle im Raum bleiben [63]. Somit eignen sich Scans ohne kontinuierliche Georeferenz für kleinere Scans, bei denen Nähe bzw. regelmäßiger Sichtkontakt zum Anker sichergestellt werden kann, zum Beispiel Zimmer, für die eine initiale Georeferenz durch das Fenster möglich ist. Eine initiale Referenz gefolgt von einer längeren Distanz, zum Beispiel um ein Hinterhaus zu scannen, wo kein StreetView vorhanden ist, ist aufgrund dieser Abweichung nicht zu empfehlen.

Die Ergebnisse aus Versuch 2 und 4 zusammen bedeuten, dass die Geospatial API nicht nur bei der Geopositionierung der Punktwolken hilft, sondern auch das auf längere Strecke fehlerbehaftete, relative Tracking von ARCore korrigiert. Außerdem erlaubt die Nutzung der API die Einordnung der Punkte in ein konsistentes Koordinatensystem, weshalb mehrere Scans am gleichen Ort ohne manuellen Aufwand automatisch zusammengefügt werden können (vgl. auch Versuch 3), was bisher nur manuell mit ARCore-Scans möglich war [6, 9]. Somit bietet die Nutzung der Geospatial API Vorteile, die über reine Georeferenzierung von Scans hinaus gehen.

### 6.3 Limitationen der Ergebnisse

Die in dieser Arbeit ermittelten Ergebnisse wurden alle mit Tests in derselben Nachbarschaft im Berliner Ortsteil Moabit ermittelt. Da VPS auf StreetView-Daten basiert, welche sich zwischen Städten und Ländern in Aktualität und Qualität stark unterscheiden, ist davon auszugehen, dass die Ergebnisse dieser Arbeit nicht direkt übertragbar sind und an anderen Orten andere Genauigkeiten gemessen werden. Dennoch haben die meisten StreetView-Daten in Deutschland ein ähnliches Alter, weshalb zumindest von ähnlichen Ergebnissen auszugehen ist.

Zudem wurden die Scans dieser Arbeit im Winter durchgeführt. Die Jahreszeit kann, insbesondere dadurch, dass Teile von Hauswänden von Blättern im Sommer verdeckt werden, auch einen messbaren Einfluss auf die Genauigkeit der Ergebnisse haben.

Alle Ergebnisse wurden mit einem Smartphone des Modells *Google Pixel 6* ermittelt. Andere Modelle können sich durch Kameraauflösung, Auflösung des Tiefenbilds, sowie in Sensorik wie Luftdrucksensor, GPS und IMU unterscheiden. Alle diese Faktoren können einen Einfluss auf die Ergebnisse haben.

In Versuch 1 wurde nur eine Fassadentextur repräsentativ untersucht. Es ist vorstellbar, dass Fassaden mit geringerer Textur weniger gute Ergebnisse und Fassaden mit ausgeprägteren optischen Features bessere Ergebnisse liefern, insbesondere, da Google selbst höhere Ungenauigkeit der Depth API bei geringer Textur erwartet [18].

In Versuch 2 wurden nur Kamerapfade entlang der Nord-Süd-Achse erstellt. Somit wurde nur die Variation entlang der Ost-West-Achse bestimmt. Es ist nicht auszuschließen, dass entlang der Nord-Süd-Achse eine andere Variation vorliegt. Im Hinblick auf die vielen möglichen Einflüsse auf die Genauigkeit der Daten (siehe oben) wäre ein direkter Vergleich zwischen Nord-Süd- und Ost-West-Achse allerdings auch nicht direkt möglich, da etwaige Ungenauigkeiten auf andere Einflüsse wie Wetter, Lichtverhältnisse oder Alter der StreetView-Bilder zurückführbar wären.

Für einen direkten Vergleich müssten signifikant mehr Messreihen durchgeführt werden, um andere Faktoren ausschließen bzw. ihren Einfluss näher quantifizieren zu können (vgl. Kapitel 8).

Für die Ergebnisse aus Versuch 4 gilt zu beachten, dass der Effekt nur für relativ weit entfernte (andere Straßenseite, ca. 16m) Hauswände getestet wurde. Die Abweichung verhält sich unter Umständen anders, wenn Objekte näher im Bild sind, bei anderer Gehgeschwindigkeit, oder Blickrichtung nach vorne oder auf den Boden. Es empfiehlt sich hier weitergehende Tests in Abhängigkeit von diesen Parametern durchzuführen (vgl. Kapitel 8).

## 7 Fazit

Im Rahmen dieser Arbeit wurde die Android-App *Urban Scanner* erstellt, welche es erlaubt georeferenzierte Punktwolken in Straßenzügen zu erheben. Hierfür wird keine externe Hardware benötigt, da die Tiefeninformationen dank der Depth API in Echtzeit auf dem Smartphone verfügbar sind und die Geospatial API präzise Ortung in Straßenzügen ermöglicht.

Die Genauigkeit der mit der App erhebbaren Punktwolken wurde bestimmt, aufgeschlüsselt nach je dem Einfluss der Depth API und der Geospatial API. Bei Scans auf größere Distanzen (z.B. von der anderen Straßenseite einer kleinen Straße, ca. 16m) kann zwar viel Fläche der Hauswand abgedeckt werden, die Depth API hat aber viel Streuung in den gefundenen Punkten, sodass Hauswände sehr „dick“ werden (1m und mehr). Nur große geometrische Features, wie zum Beispiel Balkone, gehen in dieser Streuung nicht unter. Somit ist bei Scans aus dieser Entfernung lediglich die Hauswand selbst als geometrisches Feature erkennbar, Details wie Hauseingänge, Fenster oder Verzierungen können höchstens durch die RGB-Werte, aber nicht durch Punktpositionen gefunden werden. Bei Scans aus der Nähe (< 3m) konnten die Wände deutlich genauer erkannt werden, sodass kleinere geometrische Eigenschaften wie Fenster (ca. 10cm Tiefenunterschied) dreidimensional erkennbar sind. Dafür können auf diesen Abstand nur das Erdgeschoss und vielleicht der erste Stock gescannt werden.

Die Geospatial API hat sich in den Untersuchungen dieser Arbeit als sehr genau erwiesen. Horizontale Positionsabweichungen blieben meist unter einem halben Meter, im Schnitt mit aktuellen StreetView-Daten sogar bei nur 12cm. Und selbst mit nicht-aktuellen StreetView-Daten der betrachteten Häuserfront wurde eine maximale Abweichung von ca. 1m erkannt, bei einer durchschnittlichen Abweichung von nur 32cm. Die Genauigkeitsangaben der API waren in den meisten Fällen höher als die tatsächliche Abweichung, mit vereinzelten Ausnahmen. Diese hohe Genauigkeit der Position erlaubt das Zusammenfügen mehrerer Scans, sodass ganze Straßenzüge oder Wohnblöcke in mehreren Abschnitten gescannt werden können. Außerdem sorgt diese hohe Genauigkeit dafür, dass mit ARCore großflächige Scans überhaupt möglich werden, da das relative Postionstracking von ARCore via SLAM über Distanzen > 50m sehr große Positionsfehler aggregiert.

Der Export der Daten aus der App im UTM-Koordinatensystem erlaubt eine globale Georeferenzierung, und durch manuelle Konfiguration der UTM-Zone können auch an Zonengrenzen kontinuierliche Scans erstellt werden. Die Kompatibilität des Datenformats mit *CloudCompare* ermöglicht die Umwandlung in andere gängige Formate von Punktwolken (beispielsweise .ply oder .las/.laz). Durch die Konfiguration von Schwellwerten können schlechte Punkte bereits zum Zeitpunkt des Scans ausgeschlossen werden. Die Option eines einzelnen, georeferenzierten Ankers erlaubt georeferenzierte Scans von kleinen Innenräumen oder Innenhöfen mit einer Georeferenz zu Beginn des Scans (z.B. aus einem Fenster oder vor der Haustür).

Alles in allem kann gesagt werden, dass sich die App zur Erhebung grober, georeferenzierte Punktwolken in Straßenzügen eignet. Sie ermöglicht so Zugang zur Erstellung georeferenzierte Punktwolken zu einem sehr geringen Kostenpunkt, da nur ein Android-Smartphone mit Un-

terstützung der Depth API und Geospatial API erforderlich ist. Die Position und Optik von Hauswänden und anderen, großen Features ist in diesen Wolken erkennbar. Bei Bedarf können auch detaillierte Scans von Objekten in Bodennähe (oder vom Boden selbst) erstellt werden, solange ein kurzer Scanabstand von  $< 3m$  eingehalten werden kann, um kleinere Details aufzunehmen. Die globale Position der Punktwolken ist sehr präzise bestimmt und erfordert keine manuelle Positionierung im Nachhinein. Die so erstellten Punktwolken erlauben eine Vielzahl von Anwendungsfällen, von reiner Visualisierung, über beispielsweise die Verfolgung von Bauvorhaben oder Parkplatznutzung, bis hin zur Implementation eigener visueller Positionierungsdienste, die nach einem ähnlichen Prinzip wie Googles VPS arbeiten können.

## 8 Ausblick

Die entwickelte App bietet viele Ansatzpunkte zur weiteren Entwicklung und für weitere Forschung. Zum einen ist der Scanfortschritt aktuell nicht gut für den Nutzer sichtbar (abgesehen von der Anzahl erkannter Punkte). Eine Live-Visualisierung der bisher gescannten Bereiche würde hier Abhilfe schaffen und klar anzeigen, welche Bereiche in der Punktwolke noch fehlen. Hierbei muss die Performance beachtet werden, ein simples Anzeigen aller Punkte würde die Framerate signifikant reduzieren, und damit die Qualität der Tiefenerkennung stören (siehe Kapitel 5.6). Stattdessen sollte ein Teil der Punkte angezeigt werden, oder ein simples Mesh generiert werden.

Ein weiterer Ansatzpunkt ist die Qualität der Scans. Die Depth API funktioniert zwar dank der kürzlich eingeführten Erweiterung auf 65m auf größere Distanz, hat aber eine große Streuung in den Daten, sodass erkannte Wände sehr „dick“ werden. Zur Mitigation sind viele Techniken vorstellbar, wie zum Beispiel Erkennung der Geometrie des Gebäudes und Projektion der Punkte auf diese Geometrie [40, 57], Rekonstruktion von Oberflächen und Entfernung von Rauschen [64–66] oder die Fusionierung von Tiefenbildern aus mehreren Perspektiven [41]. Auch Techniken wie die Erhöhung der Auflösung des Tiefenbilds durch Interpolation können zu besseren Ergebnissen führen [3]. Auch bestehende Ansätze zur Entfernung von Obstruktionen, wie z.B. Bäumen [41] würden die Qualität der Scans deutlich erhöhen. Viele dieser Techniken können auch durch „Qualitätsbewertungen“ der Punkte mit Confidence-Wert und Abstand zur Kamera (je weiter weg, desto „schlechter“ der Punkt) erweitert werden, um „bessere“ Punkte höher zu gewichten.

Natürlich ist auch vorstellbar, die Anforderung der Verarbeitung auf dem Smartphone zu verwerfen und auf Fotogrammetrie zu setzen, mit präziser Georeferenzierung durch die Geospatial API mittels z.B. den EXIF-Daten von Bildern. Software wie zum Beispiel *SfM-georef* [67] unterstützt die Georeferenzierung durch bekannte Kamerapositionen [68] und kann in SfM-Software wie zum Beispiel *VisualSfM* [69] genutzt werden.

Ein interessanter Ansatz wäre die Nutzung externer Datenquellen zur Validierung von Tiefeninformationen. So könnten die Geometrien von Gebäuden aus *OpenStreetMap* (OSM) genutzt werden, um nur Punkte in Scans zu behalten, die entlang dieser Hauswand verlaufen, oder Punkte auf diese Wände zu projizieren. So würde man die Geometrien aus OSM mit den Bildern des Smartphones „anmalen“ und so mit Texturen ausstatten. Die Positionen von potenziellen Obstruktionen wie z.B. Bäumen könnte auch aus OSM geladen werden, um diese automatisch zu erkennen und auszuschließen.

Die entwickelte App produziert bisher alle Punkte aus einem Tiefenbild, unabhängig von bestehenden Punkten an diesen Positionen. Dies sorgt dafür, dass bestimmte Orte immer mehr Punkte erhalten, je öfter sie im Bild sind. Alternativ kann die Dichte der Punktwolke limitiert werden, um die Datenmengen zu reduzieren. Neue Punkte könnten in diesem Fall z.B. die Confidence (oder einen neuen Score) eines Punktes erhöhen, anstatt einen weiteren an fast der gleichen Position zu erstellen.

Für die Tiefenerkennung wurde in dieser Arbeit die „Raw Depth API“ genutzt, welche genauere, aber lückenhafte Tiefendaten liefert. Die Nutzung der interpolierten Tiefe der Depth API

könnte zukünftig auch geprüft werden, da so dichtere Punktwolken generiert werden können.

Die Anwendung von unbemannten Fluggeräten (UAVs oder Drohnen) kann auch untersucht werden. Mit ihnen wäre es möglich über eine hohe vertikale Fläche einen geringen Scanabstand zu halten und damit von der hohen Scanqualität bei kleinem Scanabstand zu profitieren. Die Möglichkeit automatischer Flugbewegungen kann sicherstellen, dass die gesamte Oberfläche von einem Gebäude „abgeflogen“ und damit gescannt wird, ohne dass eine Live-Visualisierung implementiert werden muss. Für bestmögliche Abdeckung und damit mehrere Quellen zur Positionsbestimmung vorliegen, sollte die Montage mehrerer Smartphones am Fluggerät in Betracht gezogen werden.

Sehr interessant wäre es, die erhobenen, georeferenzierten Punktwolken zur Entwicklung eines eigenen, visuellen Ortungssystems ähnlich Googles VPS zu nutzen. Da die Geospatial API zur Ortung kontinuierlich mit Google Servern kommunizieren muss und damit die Position des Nutzers übermittelt, ist sie bei strengen Datenschutzvorgaben u.U. nicht anwendbar. Bei einer großen Anzahl von Nutzern entstehen auch Kosten für die Nutzung der API. Durch die Entwicklung eines eigenen Systems kann dies umgangen werden. Unter Nutzung von ARCores relativem Tracking, oder anderen, genauereren Indoor-Loaklisierungssystemen, können auch visuelle Indoor-Lokalisierungssysteme auf diese Art und Weise erstellt werden. Das andere Lokalisierungssystem muss dann nur temporär eingerichtet sein und kann durch das visuelle ersetzt werden, sobald die Daten aufgezeichnet sind.

Im Rahmen dieser Arbeit konnten nicht alle Umgebungs faktoren getestet werden. Sowohl Depth API als auch die Geospatial API werden vermutlich von Faktoren wie Scanwinkel, mehr verschiedenen Scandistanzen, unterschiedlichen Gebäudetexturen, unterschiedlichen Tages- und Jahreszeiten, Gehgeschwindigkeit, sowie Wetterlagen beeinflusst. Für all diese Variablen können weitere Tests nach dem in dieser Arbeit präsentierten Schema durchgeführt werden, um die Genauigkeit näher zu klassifizieren und optimale Scanbedingungen zu bestimmen.

Nähere Tests zum Positionsfehler von ARCores relativem Positionstracking könnten Wege identifizieren, um diesen Fehler zu korrigieren. Die in dieser Arbeit identifizierte Abhängigkeit vom Blickwinkel legt nahe, dass der Blickwinkel direkt den Fehler beeinflusst. Interessant wären hier die Fehlerbilder bei unterschiedlichen Winkeln zur Gehrichtung (hier wurden nur 90° getestet), unterschiedlicher Gehgeschwindigkeit, unterschiedlichen Geräten und unterschiedlichen Distanzen von Objekten zur Kamera.

## 9 Literaturverzeichnis

- [1] J. Valentin u. a., „Depth from Motion for Smartphone AR,“ *ACM Trans. Graph.*, Vol. 37, Nr. 6, Dez. 2018, ISSN: 0730-0301. DOI: 10.1145/3272127.3275041. Adresse: <https://doi.org/10.1145/3272127.3275041>.
- [2] Google, Inc., *May 2022 (ARCore SDK version 1.31) changes to Depth*, Englisch, Mai 2022. Adresse: <https://developers.google.com/ar/develop/depth/changes> (besucht am 12.03.2023).
- [3] R. Haenel, Q. Semler, E. Semin, P. Grussenmeyer und E. Alby, „Integration of Depth Maps from Arcore to Process Point Clouds in Real Time on a Smartphone,“ in *XXIV ISPRS Congress, 5-9 juillet 2021, Nice (en ligne), France*, Bd. 43, 2021, S. 201–208.
- [4] G. Luetzenburg, A. Kroon und A. A. Bjørk, „Evaluation of the Apple iPhone 12 Pro LiDAR for an application in geosciences,“ *Scientific reports*, Vol. 11, Nr. 1, S. 22221, 2021.
- [5] L. Vonásek, *3D Live Scanner*, Englisch/Deutsch, 2022. Adresse: <https://play.google.com/store/apps/details?id=com.lvonasek.arcore3dscanner> (besucht am 24.02.2023).
- [6] D. Costantino, G. Vozza, M. Pepe und V. S. Alfio, „Smartphone LiDAR Technologies for Surveying and Reality Modelling in Urban Scenarios: Evaluation Methods, Performance and Challenges,“ *Applied System Innovation*, Vol. 5, Nr. 4, S. 63, 2022.
- [7] Google, Inc., *Build global-scale, immersive, location-based AR experiences with the ARCore Geospatial API*, Englisch. Adresse: <https://developers.google.com/ar/develop/geospatial> (besucht am 24.02.2023).
- [8] S. Harwin und A. Lucieer, „Assessing the accuracy of georeferenced point clouds produced via multi-view stereopsis from unmanned aerial vehicle (UAV) imagery,“ *Remote Sensing*, Vol. 4, Nr. 6, S. 1573–1599, 2012.
- [9] J.-A. Paffenholz, „Direct geo-referencing of 3D point clouds with 3D positioning sensors,“ 2012.
- [10] Google, Inc., *Obtain the device camera's Geospatial pose (Webarchiv)*, Englisch, Dez. 2022. Adresse: <https://web.archive.org/web/20221206193249/https://developers.google.com/ar/develop/java/geospatial/obtain-device-pose> (besucht am 24.02.2023).
- [11] W. Li u. a., „AADS: Augmented autonomous driving simulation using data-driven algorithms,“ *Science robotics*, Vol. 4, Nr. 28, eaaw0863, 2019.
- [12] T. Mikita, M. Balková, A. Bajer, M. Cibulka und Z. Patočka, „Comparison of different remote sensing methods for 3d modeling of small rock outcrops,“ *Sensors*, Vol. 20, Nr. 6, S. 1663, 2020.

- [13] C. Gollob, T. Ritter, R. Kraßnitzer, A. Tockner und A. Nothdurft, „Measurement of forest inventory parameters with Apple iPad Pro and integrated LiDAR technology,“ *Remote Sensing*, Vol. 13, Nr. 16, S. 3129, 2021.
- [14] D. Du u. a., „The unmanned aerial vehicle benchmark: Object detection and tracking,“ in *Proceedings of the European conference on computer vision (ECCV)*, 2018, S. 370–386.
- [15] Google, Inc., *Video: What's New on Tango (Google I/O '17)*, Englisch, Mai 2017. Adresse: <https://www.youtube.com/watch?v=B0rg2oc3-rQ> (besucht am 13.03.2023).
- [16] P. France, *Video: Using 3D Scans to Make an Adorable Hiking Render*, Englisch, Juli 2022. Adresse: <https://www.youtube.com/watch?v=FP0bYDkoLWc> (besucht am 11.03.2023).
- [17] SarahC, *Reddit-Beitrag: Mixing NeRF + Photogrammetry: The result!* Englisch, Sep. 2022. Adresse: [https://www.reddit.com/r/photogrammetry/comments/xo9biz/mixing\\_nerf\\_photogrammetry\\_the\\_result/](https://www.reddit.com/r/photogrammetry/comments/xo9biz/mixing_nerf_photogrammetry_the_result/) (besucht am 11.03.2023).
- [18] Google, Inc., *ARCore Documentation: Depth adds realism*, Englisch. Adresse: <https://developers.google.com/ar/develop/depth> (besucht am 28.02.2023).
- [19] F. Yilmazturk und A. E. Gurbak, „Geometric evaluation of mobile-phone camera images for 3D information,“ *International Journal of Optics*, Vol. 2019, S. 1–10, 2019.
- [20] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi und R. Ng, „NeRF: Representing scenes as neural radiance fields for view synthesis,“ *Communications of the ACM*, Vol. 65, Nr. 1, S. 99–106, 2021.
- [21] R. Paharia, *A new wave of AR Realism with the ARCore Depth API*, Englisch, Juni 2020. Adresse: <https://developers.googleblog.com/2020/06/a-new-wave-of-ar-realism-with-arcore-depth-api.html> (besucht am 12.03.2023).
- [22] Google, Inc., *Use Raw Depth in your Android app*, Englisch. Adresse: <https://developers.google.com/ar/develop/java/depth/raw-depth> (besucht am 12.03.2023).
- [23] Google, Inc., *ARCore Dokumentation: Frame.acquireRawDepthImage16Bits()*, Englisch. Adresse: <https://developers.google.com/ar/reference/java/com/google/ar/core/Frame#acquireRawDepthImage16Bits-> (besucht am 11.12.2022).
- [24] Google, Inc., *ARCore supported devices*, Englisch, Feb. 2023. Adresse: <https://developers.google.com/ar/devices> (besucht am 01.03.2023).
- [25] B. Sidhu und E. Lai, *Make the world your canvas with the ARCore Geospatial API*, Englisch, Mai 2022. Adresse: <https://developers.googleblog.com/2022/05/Make-the-world-your-canvas-ARCore-Geospatial-API.html> (besucht am 13.03.2023).
- [26] Twitterprofil, *Dereck Bridie (@devbridie)*, Englisch. Adresse: <https://twitter.com/devbridie> (besucht am 13.03.2023).

- [27] Google, Inc., *Check VPS availability at the device's current location*, Englisch. Adresse: <https://developers.google.com/ar/develop/java/geospatial/check-vps-availability> (besucht am 13.03.2023).
- [28] J. W. Hager, J. F. Behensky und B. W. Drew, „The universal grids: Universal transverse mercator (UTM) and universal polar stereographic (UPS). Edition 1,“ Defense Mapping Agency Hydrographic/Topographic Center Washington Dc, Techn. Ber., 1989.
- [29] E. Turner und Google, Inc., *Codelab ARCore Raw Depth*, Englisch, Mai 2022. Adresse: <https://codelabs.developers.google.com/codelabs/arcore-rawdepthapi> (besucht am 16.02.2023).
- [30] Google, Inc., *Obtain the device camera's Geospatial pose*, Englisch. Adresse: <https://developers.google.com/ar/develop/java/geospatial/obtain-device-pose> (besucht am 14.03.2023).
- [31] The MathWorks, Inc., *MATLAB Dokumentation: earthRadius*, Englisch, Okt. 2010. Adresse: <https://www.mathworks.com/help/map/ref/earthradius.html> (besucht am 14.03.2023).
- [32] D. Tsoukalos, V. Drosos und D. K. Tsolis, „Attempting to reconstruct a 3D indoor space scene with a mobile device using ARCore,“ in *2021 12th International Conference on Information, Intelligence, Systems & Applications (IISA)*, 2021, S. 1–6. DOI: 10.1109/IISA52424.2021.9555529.
- [33] SmartMobileVision, *SCANN3D*, Englisch, Dez. 2017. Adresse: <https://play.google.com/store/apps/details?id=com.smartmobilevision.scann3d> (besucht am 28.02.2023).
- [34] A. Sprefaico, F. Chiabrando, L. Teppati Losè und F. Giulio Tonolo, „The ipad pro built-in lidar sensor: 3d rapid mapping tests and quality assessment,“ *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, Vol. 43, S. 63–69, 2021.
- [35] Capturing Reality und Epic Games, Inc., *RealityScan is now free to download on iOS*, Englisch, Dez. 2022. Adresse: <https://www.unrealengine.com/en-US/blog/realityscan-is-now-free-to-download-on-ios> (besucht am 28.02.2023).
- [36] Polycam, *polycam Homepage*, Englisch, Dez. 2022. Adresse: <https://poly.cam/> (besucht am 28.02.2023).
- [37] Laan Labs, *3d Scanner App*, Englisch, Feb. 2022. Adresse: <https://apps.apple.com/de/app/3d-scanner-app/id1419913995> (besucht am 28.02.2023).
- [38] Luma AI, Inc., *Luma AI iOS App*, Englisch, Feb. 2023. Adresse: <https://apps.apple.com/us/app/luma-ai/id1615849914> (besucht am 01.03.2023).
- [39] Luma AI, Inc., *Luma AI Website*, Englisch. Adresse: <https://lumalabs.ai/> (besucht am 01.03.2023).

- [40] J. Xiao, T. Fang, P. Zhao, M. Lhuillier und L. Quan, „Image-Based Street-Side City Modeling,“ *ACM Trans. Graph.*, Vol. 28, Nr. 5, S. 1–12, Dez. 2009, ISSN: 0730-0301. DOI: 10.1145/1618452.1618460. Adresse: <https://doi.org/10.1145/1618452.1618460>.
- [41] M. Pollefeys u. a., „Detailed real-time urban 3d reconstruction from video,“ *International Journal of Computer Vision*, Vol. 78, S. 143–167, 2008.
- [42] M. Tancik u. a., „Block-NeRF: Scalable Large Scene Neural View Synthesis,“ *arXiv*, 2022.
- [43] DMA WGS 84 Development Committee, *World Geodetic System 1984 - Its Definition and Relationships with Local Geodetic Systems*. The Defense Mapping Agency, 1984.
- [44] J. Minor, *Android: Googles Betriebssystem verliert in Deutschland immer mehr Marktanteile – liegt nur noch bei 67 Prozent*, Deutsch, Juni 2022. Adresse: <https://www.googlewatchblog.de/2022/06/android-googles-betriebssystem-deutschland/> (besucht am 01.03.2023).
- [45] Google, Inc. und Open Geospatial Consortium, Inc., *Keyhole Markup Language*, Englisch. Adresse: <https://developers.google.com/kml> (besucht am 01.03.2023).
- [46] Google, Inc., *My Maps*, Deutsch. Adresse: [https://www.google.com/intl/de\\_de/maps/about/mymaps/](https://www.google.com/intl/de_de/maps/about/mymaps/) (besucht am 01.03.2023).
- [47] Z. Huang, Y. Wen, Z. Wang, J. Ren und K. Jia, „Surface reconstruction from point clouds: A survey and a benchmark,“ *arXiv preprint arXiv:2205.02413*, 2022.
- [48] Google, Inc., *Sharing a file*, Englisch. Adresse: <https://developer.android.com/training/secure-file-sharing/share-file> (besucht am 01.03.2023).
- [49] Google, Inc., *Geospatial quickstart for Android*, Englisch. Adresse: <https://developers.google.com/ar/develop/java/geospatial/quickstart> (besucht am 19.03.2023).
- [50] User: kaj777, D. Bridie und E. Maggio, *GitHub issue: Depth Resolution mismatch*, Englisch, Nov. 2021. Adresse: <https://github.com/google-ar/arcore-android-sdk/issues/1314> (besucht am 15.03.2023).
- [51] D. Girardeau-Montaut, *CloudCompare - 3D point cloud and mesh processing software - Open Source Project*, Englisch. Adresse: <https://www.danielgm.net/cc/> (besucht am 16.03.2023).
- [52] P. Abeles, *BoofCV Main Page*, Englisch. Adresse: [https://boofcv.org/index.php?title=Main\\_Page](https://boofcv.org/index.php?title=Main_Page) (besucht am 16.03.2023).
- [53] Google, Inc., *ARCore Dokumentation: Converting coordinates between camera images and depth images*, Englisch. Adresse: [https://developers.google.com/ar/develop/java/depth/developer-guide#converting\\_coordinates\\_between\\_camera\\_images\\_and\\_depth\\_images](https://developers.google.com/ar/develop/java/depth/developer-guide#converting_coordinates_between_camera_images_and_depth_images) (besucht am 16.03.2023).
- [54] OrbisGIS und M. Michaud, *GitHub: Coordinate Transformation Suite*, Englisch, Jan. 2022. Adresse: <https://github.com/orbisgis/cts> (besucht am 17.03.2023).

- [55] Kotlin Foundation, *Kotlin Dokumentation: Coroutines*, Englisch. Adresse: <https://kotlinlang.org/docs/coroutines-overview.html> (besucht am 17.03.2023).
- [56] Kotlin Foundation, *Kotlin Dokumentation: Asynchronous Flow*, Englisch. Adresse: <https://kotlinlang.org/docs/flow.html> (besucht am 17.03.2023).
- [57] T. Hackel, J. D. Wegner und K. Schindler, „Contour detection in unstructured 3D point clouds,“ in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, S. 1610–1618.
- [58] M. Weinmann u. a., *Reconstruction and analysis of 3D scenes*. Springer, 2016, Bd. 1.
- [59] M. Pauly, M. Gross und L. P. Kobbelt, „Efficient simplification of point-sampled surfaces,“ in *IEEE Visualization, 2002. VIS 2002.*, IEEE, 2002, S. 163–170.
- [60] T. Feigl, A. Porada, S. Steiner, C. Löffler, C. Mutschler und M. Philippsen, „Localization Limitations of ARCore, ARKit, and Hololens in Dynamic Large-scale Industry Environments,“ in *VISIGRAPP (1: GRAPP)*, 2020, S. 307–318.
- [61] K. Merry und P. Bettinger, „Smartphone GPS accuracy study in an urban environment,“ *PloS one*, Vol. 14, Nr. 7, 2019.
- [62] Y. Morhenn, „Entwicklung und Evaluation Eines Mikrocontroller-gestützten Systems Zur Verbesserung Mobiler Ortung Mittels Differentiellem GPS,“ 2018.
- [63] Google, Inc., *Working with Anchors*, Englisch. Adresse: <https://developers.google.com/ar/develop/anchors> (besucht am 09.03.2023).
- [64] M. A. Fischler und R. C. Bolles, „Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography,“ *Commun. ACM*, Vol. 24, Nr. 6, S. 381–395, Juni 1981, ISSN: 0001-0782. DOI: [10.1145/358669.358692](https://doi.org/10.1145/358669.358692). Adresse: <https://doi.org/10.1145/358669.358692>.
- [65] P. Erler, P. Guerrero, S. Ohrhallinger, N. J. Mitra und M. Wimmer, „Points2surf learning implicit surfaces from point clouds,“ in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part V*, Springer, 2020, S. 108–124.
- [66] C. C. Jia, C. J. Wang, T. Yang, B. H. Fan und F. G. He, „A 3D point cloud filtering algorithm based on surface variation factor classification,“ *Procedia Computer Science*, Vol. 154, S. 54–61, 2019.
- [67] M. R. James und S. Robson, „Straightforward reconstruction of 3D surfaces and topography with a camera: Accuracy and geoscience application,“ *Journal of Geophysical Research: Earth Surface*, Vol. 117, Nr. F3, 2012.
- [68] M. James, *SfM\_georef - georeferencing SfM point clouds*, Englisch. Adresse: [https://www.lancaster.ac.uk/staff/jamesm/software/sfm\\_georef.htm](https://www.lancaster.ac.uk/staff/jamesm/software/sfm_georef.htm) (besucht am 19.03.2023).

- [69] C. Wu, *VisualSFM : A Visual Structure from Motion System*, Englisch. Adresse: <http://ccwu.me/vsfm/> (besucht am 19.03.2023).
- [70] Google, Inc., *Geospatial anchors*, Englisch. Adresse: <https://developers.google.com/ar/develop/c/geospatial/geospatial-anchors> (besucht am 12.02.2023).

## 10 Abkürzungsverzeichnis

<b>API:</b>	Application Programming Interface
<b>AR:</b>	Augmented Reality
<b>CSV:</b>	Comma-separated values
<b>DGPS:</b>	Differential Global Positioning System
<b>EPSG:</b>	European Petroleum Survey Group Geodesy
<b>EXIF:</b>	Exchangeable Image File Format
<b>FPS:</b>	Frames per second
<b>GPS:</b>	Global Positioning System
<b>IMU:</b>	Inertial Measurement Unit
<b>INS:</b>	Inertial Navigation System
<b>KML:</b>	Keyhole Markup Language
<b>LiDAR:</b>	Light Detection And Ranging
<b>NeRF:</b>	Neural Radiance Field
<b>RGB:</b>	Rot-Grün-Blau
<b>SfM:</b>	Structure-from-Motion
<b>SLAM:</b>	Simultaneous Localization And Mapping
<b>ToF:</b>	Time of Flight
<b>UAV:</b>	Unmanned Aerial Vehicle
<b>UTM:</b>	Universal Transverse Mercator
<b>VPS:</b>	Visual Positioning Service

## 11 Abbildungsverzeichnis

1	Beispiel für ein Tiefenbild (rechts) neben dem normalen Bild (links). Die Tiefenwerte sind durch eine Farbskala visualisiert. Übernommen aus [18]. . . . .	3
2	Schematische Zeichnung der Berechnung von $p_x$ aus den intrinsischen Kamera-parametern. Das Koordinatensystem der Kamera ist dargestellt mit Achsen und Ursprung. Die Größe $p_x$ ist die X-Koordinate des Punktes im Koordinatensystem der Kamera, $c_x$ die X-Komponente des Bildmittelpunktes, $f_x$ die Brennweite in Pixeln, $d$ der Tiefenwert für den untersuchten Pixel, $t$ der aktuell untersuchte Punkt in der realen Welt und $t'_x$ die X-Koordinate vom Bild des aktuell untersuchten Punktes auf dem Schirm. Es gilt zu beachten, dass das Bild auf dem Schirm nicht gespiegelt ist, im Gegensatz zu einer normalen Lochkamera. . . . .	7
3	Ansichten der App nach erstem App-Start . . . . .	17
4	Detailansicht einer Scankonfiguration . . . . .	18
5	Variationen des Status-Indikators . . . . .	20
6	Verschiedene Scanansichten . . . . .	21
7	Konfigurationsmöglichkeiten des UTM-Postprocessors . . . . .	22

8	Dialog für Verschiebung des Koordinatensystems beim Import der UTM-Punktwolke in CloudCompare . . . . .	25
9	Filteroption in CloudCompare visualisiert anhand eines Scans der App . . . . .	26
10	Schematische Darstellung der wichtigsten Schritte im Scanprozess bei der Verarbeitung von Frames . . . . .	27
11	Konfiguration und Ausführung des UTM Postprocessors . . . . .	31
12	Die untersuchte Hauswand. Der untersuchte Wandbereich wurde rot markiert und ist ca. 90cm breit, 2,20m hoch und steht 10cm vom Rest der Wand hervor. . . . .	38
13	Wand (orange) und Kamerapfade (grün) für die Scans aus unterschiedlichen Entfernungen (Aufsicht). Eine Kameraposition ist beispielhaft mit der Bewegungsrichtung und dem Sichtkegel gekennzeichnet. . . . .	38
14	Auswertung des Abstands (rot) zwischen erkannten Wandpunkten (blau) und der Ebene des Startpunkts des Kamerapfads (oliv). Die Punkte des Kamerapfads sind grün gekennzeichnet. (Aufsicht) . . . . .	39
15	Histogramme der Planarität und Oberflächenvarianz für Scanabstände zwischen 1m und 6m . . . . .	41
16	Histogramme der Planarität und Oberflächenvarianz für Scanabstände zwischen 8m und 16m . . . . .	42
17	Boxplot-Diagramm der Punktestreuung für Scans aus verschiedenen Distanzen. Es ist jeweils die Differenz von Depth-API-Abstand und realem Abstand gegeben. Ausreißer sind solche Werte, die weiter als das 1,5-fache vom Interquartilabstand über dem dritten oder unter dem ersten Quartil liegen. . . . .	43
18	Scanqualität aus den Entfernungen 1m bis 16m bei der gleichen Wand. . . . .	44
19	Vergleich zwischen der StreetView-Ansicht und der aktuellen Häuserfront der „Teststrecke Süd“. Der Autohändler vom StreetView-Bild wurde durch ein Wohnhaus ersetzt. . . . .	45
20	Auswertung der Positionsabweichung (rot) des Kamerapfads (grün) von der Ebene zwischen Start- und Endpunkt (blau). Beim Test wurde eine gerade Linie zwischen Start- und Endpunkt abgelaufen, weshalb die Ebene entlang des „Soll-Pfads“ verläuft und die roten Abstände die Positionsabweichung darstellen. (Aufsicht) .	46
21	Diagramme zur horizontalen Genauigkeit der Geospatial API an zwei Teststrecken, eine ohne neue Bebauung seit der Aufnahme der StreetView-Bilder („Teststrecke Nord“, links) und eine mit neuer Bebauung („Teststrecke Süd“, rechts). Die x-Achse ist für Teststrecke Süd invertiert, da die x-Koordinate im Laufe des Tests kleiner wurde. . . . .	47
22	Ansichten der Scans aus 3m und 16m Entfernung. Es ist der Einfluss des Baumes (Abbildung 22b, links im Bild, steht aus der Wand hervor), sowie der Vergleich der Wanddicke zum Scan aus 3m Entfernung (Abbildungen 22c und 22d) sichtbar.	51

23	Kamerapfade ohne kontinuierliche Georeferenz (mit Tracking relativ zu einem initialen Anker) . . . . .	53
24	Aufsicht auf die Streuung der Confidence-Werte (rot = hohe Confidence, blau = niedrige Confidence). Vor einer Wand (unten im Bild) liegen Ausreißer mit sehr hoher Confidence (oben im Bild) und in der Wand sind die Confidence-Werte stark gestreut. Der Maßstab ist in Metern angegeben. . . . .	55
25	Test für Referenz der in der Depth API angegebenen Tiefe. . . . .	56
26	Unterschiedliche Interpretationen der Tiefe eines Objekts, entweder zur Linse oder zur Linsenebene (Aufsicht) . . . . .	56
27	Kombination aus Indoor- und Outdoor-Scans . . . . .	57
28	Höhe eines Indoor Scans (blau) ist nicht korrekt (sollte eigentlich auf einer Höhe mit dem roten Scan sein) . . . . .	58

## 12 Glossar

**Confidence:** Ein Wert zugehörig zu einem Pixel im Tiefenbild. Dieser gibt an, wie „sicher“ die Tiefenerkennung bei der angegebenen Tiefe für den Punkt ist.

**Depth API:** API von ARCore, welche die Tiefe von Pixeln im Bild ermittelt, primär nur durch die Bilder einer bewegten Kamera.

**EPSG-Codes:** Ein Codesystem zur eindeutigen Identifizierung von Koordinatenreferenzsystemen. Jede UTM-Zone besitzt einen eigenen EPSG-Code.

**Georeferenzierung:** Einer Punktwolke oder Geometrie globale Koordinaten, also die Position auf der Erde, zuweisen.

**Geospatial Anchor:** Ein Anker aus ARCore, welcher in Abhängigkeit von seiner Position auf der Erde erstellt wird. Die Position dieser Anker wird kontinuierlich in Abhängigkeit von der aktuell bestimmten Gerätelocation angepasst [70].

**Geospatial API:** API von ARCore, die es ermöglicht die globale Position des Smartphones zu ermitteln. Die Position aus dieser API kann von GPS oder vom Visual Positioning Service (VPS) stammen.

**Pose:** Eine Pose ist zusammengesetzt aus einer Position und einer Rotation (häufig auch eine Skalierung, aber dies ist in dieser Arbeit nicht relevant).

**Postprocessor:** Ein Vorgang zur Nachverarbeitung von Scans, siehe Kapitel 4.4.1, 4.1.6

und 4.1.7.

**Scan:** Bezeichnung für den Vorgang zur Erhebung von Punktwolken im Rahmen dieser Arbeit.

**Szenenkoordinatensystem:** Das Koordinatensystem der ARCore Szene.

**Visual Positioning Service:** Cloud-Service von Google zur visuellen Positionierung eines Smartphones.

## **Eigenständigkeitserklärung**

Hiermit versichere ich an Eides statt durch meine Unterschrift, dass ich die vorstehende Arbeit selbstständig und ohne fremde Hilfe angefertigt und alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe, mich auch keiner anderen als der angegebenen Literatur oder sonstiger Hilfsmittel bedient habe. Die Arbeit hat in dieser oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

Berlin, den 21. März 2023



Emil Schoenawa